GROWING STRUCTURED MEMORY WITH LEARNED LOCAL DECISIONS FOR ONLINE INGESTION

Anonymous authors
Paper under double-blind review

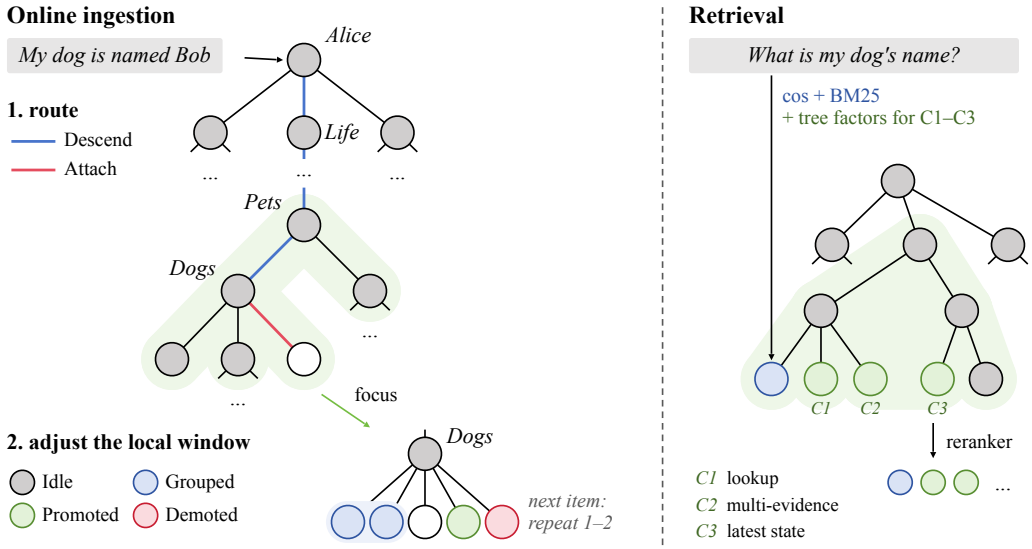

Figure 1: How the memory is built and used. **Left:** each arriving item is routed from the root to a leaf, choosing at every internal node between stopping there and descending into a child category; the bounded window around where it lands is then revised, each item in it either staying, moving up, sinking into a sibling category, or joining others in founding a new one. Nothing outside that window is read or rewritten, and the same 0.5B policy network makes both decisions. Repeating this grows the hierarchy one item at a time. **Right:** a query is answered by searching the resulting hierarchy, and what comes back are leaves, which are the original items rather than summaries of them.

ABSTRACT

In online memory construction, a corpus arrives incrementally and each new item must be ingested upon arrival. Explicit structure over items, such as knowledge graphs, can substantially improve retrieval, but growing and maintaining such structure under incremental arrival remains a challenge. Recent methods address this agentically, prompting an off-the-shelf LLM to perform each ingestion; this yields flexible updates, but the cost of these calls is high and grows with corpus size. We instead grow a structured memory through learned local decisions, coupling a hierarchy over the corpus, here a tree, with a model trained to maintain it and a retrieval procedure that exploits it. We cast ingestion as a local maintenance problem: each decision depends only on a bounded region of the tree, not on the memory as a whole. A 0.5B model, supervised-finetuned on decisions labeled offline by a larger model, is enough to make these decisions. Retrieval in turn draws on the hierarchy as a signal beyond flat similarity, recovering the benefit of structure without any LLM calls. We evaluate on standard benchmarks under both offline and online arrival, where it matches state-of-the-art memory systems in R@10 at a fraction of their ingestion cost. On CloneMem, RealMem and NovelQA it reaches R@10 of 35.70, 71.46 and 60.67, against 32.76, 67.86 and 57.56 for the strongest competing system, while spending less than a tenth of

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

the hosted-LLM input tokens of the cheapest competitor and building its memory at least eight times faster.

1 INTRODUCTION

Many systems accumulate information over a long horizon, such as LLM assistants and copilots that need to retain user context across sessions, personal knowledge bases that grow as notes are added, and document collections indexed as they are written. In each of them the information arrives online, and organizing it into a searchable memory is a problem in its own right. For agents in particular, such a memory has become a standard component of the stack (Packer et al., 2023; Zhong et al., 2023). The corpus arrives incrementally, and each new item must be ingested upon arrival, because a query may reach it before any later item exists. Ingestion is therefore not a preprocessing step that can be amortized offline: it is an operation the system performs throughout its lifetime, once per item. How that operation organizes what it ingests is consequently doing much of the work—it settles what later retrieval is able to reach, and its per-item price is paid again every time the memory grows.

Explicit structure over the items improves retrieval, and a substantial body of work builds memories of this kind (Edge et al., 2024; Gutiérrez et al., 2025; Rasmussen et al., 2025; Sarthi et al., 2024; Rezazadeh et al., 2024). Maintaining such a structure online is a different problem from constructing one in batch. Each arriving item raises the same question—where does it belong, and what does its arrival imply for the items already around it—and answering it well is difficult for two reasons at once. The decision has to be accurate: placing an item correctly means judging how it relates in meaning to what is already stored, and a poor placement is not a local blemish, since it also shapes where later items are placed and what a query can later reach. At the same time it has to be fast, because the memory is typically searched while it is still growing, so an item must be ready to be retrieved shortly after it arrives. Accuracy of this kind is what large models are good at, but consulting one for every item is what makes it slow.

Recent memory systems answer this question with a large model, and they answer it well. A-Mem (Xu et al., 2025) has an LLM write an atomic note for each arriving memory and link it to the notes it belongs with; Mem0 (Chhikara et al., 2025) extracts the salient facts and decides per fact whether to add, update or delete; MemOS (Li et al., 2025) schedules heterogeneous memory as typed units. Together these systems cover the whole path—organizing what arrives, maintaining it as more arrives, and searching it—and they report strong results. They do, however, share some limitations. Each ingestion decision is an LLM call, so organizing a corpus costs many of them, and the tokens and wall-clock time spent grow with how much has been written. Some of them also keep a flat organization: A-Mem links notes to each other and Mem0 keeps a set of facts, so neither records which items are contained in which, and a query cannot be narrowed from a general region to a specific one. HippoRAG 2 (Gutiérrez et al., 2025) makes the complementary trade: its knowledge graph supports incremental insertion and retrieves strongly by propagating relevance from query-linked seeds, but that propagation is computed over the whole graph at query time, so the work per query grows with everything written so far. In both cases the price of a good organization is paid again and again, and it scales with the size of the memory rather than with the item being ingested. On our three benchmarks, these four systems spend 11.65M to 769.64M hosted-LLM input tokens over ingestion and retrieval, and building their memory on three buckets of a benchmark takes them 4,543 to 47,797 s of serial wall-clock time (Table 1).

We instead grow a structured memory through *learned local decisions*, coupling three things (Figure 1). The first is the structure: a hierarchy over the corpus, here a tree, whose leaves each carry one item and whose internal nodes are natural-language categories nested level by level, so that containment is explicit. The second is a model trained to maintain it. Each arriving item is handled in one round of two decisions. It is first routed from the root downwards, choosing at each node between stopping there and descending into one of its child categories, until it is placed as a leaf. Its arrival then unsettles the region it landed in, and the model revises that region, deciding for each item nearby whether it stays where it is, moves up to the level above, sinks into a neighbouring category, or joins others in founding a new category of their own. Suppose the tree already organizes a user's items under categories such as *hobbies* and *career*, and "runs a weekly pottery class" arrives: it is routed into *hobbies*, and the second decision may then group it with two other craft items into a new

crafts category beneath it. The third is a retrieval procedure that reads the resulting hierarchy, rather than treating its leaves as a flat collection.

Each decision reads only a bounded region of the tree rather than the memory as a whole, so the cost of making it stays controlled and does not blow up as the memory grows. What it still needs is the semantic judgment that a large model supplies, and we obtain that by paying for it once, offline, instead of at every ingestion. A frozen 27B teacher is run over real corpora to label both decisions, and a **0.5B policy network** initialized from Qwen2.5-0.5B-Instruct is supervised-finetuned on the result; it serves both stages, selecting between them from the grammar of its own output. At deployment the policy is what maintains the memory: for each arriving item it emits the routing decision and the edits to the region around it, and every edit it emits is a legal one. Retrieval then draws on the resulting hierarchy as a signal beyond flat similarity, through a deterministic procedure that makes no generative call at query time.

We evaluate the system end to end as a retrieval task on public benchmarks, under both offline and online arrival, measuring Top-10 recall of gold evidence against Mem0, A-Mem, MemOS and HippoRAG2 on identical source units, retrieval budget, and scorer. Our system reaches R@10 of 35.70, 71.46 and 60.67 on CloneMem, RealMem and NovelQA, against 32.76, 67.86 and 57.56 for the strongest of them, HippoRAG 2, while spending less than a tenth of the hosted-LLM input tokens of the cheapest and building its memory at least eight times faster than any of them.

2 Related Work

Unstructured memory. Many systems keep memory as a collection of natural-language items retrieved by similarity. MemGPT (Packer et al., 2023) pages facts between the context window and external storage; Mem0 (Chhikara et al., 2025) extracts salient facts and decides ADD/UPDATE/DELETE per fact; MemoryBank (Zhong et al., 2023) and Generative Agents (Park et al., 2023) maintain a memory stream through summarization and recency–importance scoring; and A-Mem (Xu et al., 2025) has an LLM write and link atomic notes as each memory arrives. They show that an LLM on the write path decides well what to keep, and they ingest material as it arrives, as we do.

Graph-structured and hierarchical memory. Making the relations among items explicit improves what a query reaches. GraphRAG (Edge et al., 2024) indexes an entity graph, HippoRAG (Gutiérrez et al., 2024) and HippoRAG 2 (Gutiérrez et al., 2025) retrieve over an OpenIE knowledge graph by personalized PageRank and support incremental insertion, Zep (Rasmussen et al., 2025) maintains a bi-temporal knowledge graph, and MemOS (Li et al., 2025) schedules typed memory units. Hierarchies record containment, so a query can be narrowed level by level: RAPTOR (Sarthi et al., 2024) builds a summary tree by recursive clustering, MemTree (Rezazadeh et al., 2024), the closest to our setting, grows one online through similarity rules with depth-adaptive thresholds, and SegTree (Liu et al., 2026b), H-MEM (Sun & Zeng, 2025) and MemForest (Chen et al., 2026) add temporal and LLM-assigned indexing. Unlike these works, we *learn* the maintenance decisions themselves.

Learned memory management. Memory-R1 (Yan et al., 2025) trains ADD/UPDATE/DELETE/NOOP decisions with reinforcement learning, Mem-α (Wang et al., 2025b) learns construction policies from downstream rewards, and AgeMem (Yu et al., 2026) folds memory operations into the agent's own policy. Other work lowers the cost of maintenance through consolidation and compression (Fang et al., 2025), small language models under a fixed budget (Zhang et al., 2026), distilled context compression (Kang et al., 2025) and structured agent distillation, which preserves large-model decisions in small students (Liu et al., 2026a). These systems organize their stores by rules or slots; we maintain a hierarchy online with a small student trained on the decisions of a frozen teacher, a direction a recent survey notes is little explored for structured memory (Zhang et al., 2025).

3 Method

We organize memory as a tree over the corpus, and the online setting makes maintaining it a streaming computation. Items arrive one at a time, and each is consumed as it arrives, in a *round* of two

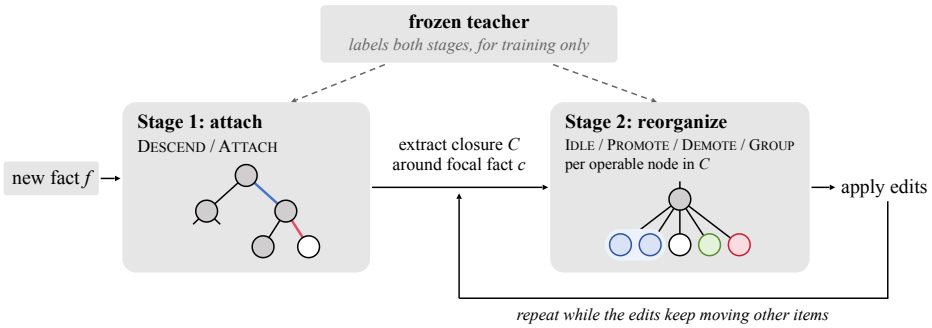

Figure 2: How one item is ingested. Stage 1 routes the arriving item from the root down to a leaf, choosing at each node between attaching there and descending into a child category. Stage 2 then revises the closure around it: each item in the window stays, moves up, sinks into a sibling category, or joins a new category. Stage 2 is repeated on the same window while its edits keep moving other items, and stops as soon as a pass changes nothing, the new item itself moves, or a GROUP founds a new category; the next item then starts at Stage 1. The frozen teacher takes the policy's place only when training labels are produced.

local decisions (§3.1): the item is routed down to a place in the tree, and the closure around where it landed is then revised so the organization stays sound as it grows. Both decisions are made by a **0.5B policy network**, trained offline on decisions labelled by a frozen teacher. Nothing is buffered until the corpus is complete and nothing is rebuilt globally. A large model is still called at deployment, but only to name the category a GROUP founds, which is a small fraction of all operations; every routing and revision decision is made by the policy. LLM calls therefore become far fewer than in systems that make one per ingestion decision, and the saving shows directly in the tokens consumed (Table 1).

Each round uses two sets of actions. In Stage 1 the policy walks down from the root, choosing at each node between ATTACH, which places the item there as a new leaf, and DESCEND into one of the node's child categories. In Stage 2 the new item becomes the *focal item* c, the closure around it is extracted, and the policy assigns every item in that window one edit: IDLE leaves it in place, PROMOTE lifts it to the grandparent, DEMOTE sinks it into a sibling category, and GROUP moves it, together with the other items marked GROUP, into one new category. The edits are applied and Stage 2 repeats on the same focal item until a pass changes nothing, the focal item itself moves, or a GROUP founds a new category, so that at most one GROUP is applied per item. Figure 2 traces one round end to end, and Algorithm 1 in the appendix states it in full.

3.1 TWO LOCAL DECISIONS

Setup. An *entity* e (a user, an institution, a project) is described by an unordered stream of atomic natural-language items $\mathcal{F}_e = \{f_1, \ldots, f_n\}$; we assume no pre-existing structure over $\mathcal{F}_e$. The task is to organize this stream, online, into a rooted tree T whose leaves carry the items and whose internal nodes carry category descriptions, the root being the entity itself. Parent–child edges make containment explicit, so related items live under a shared category, redundancy is exposed among siblings, and retrieval can narrow coarse-to-fine. Starting from a tree containing only the root, facts arrive one at a time and each triggers one round. We write $\pi^{(1)}$ for the attach decision and $\pi^{(2)}$ for the reorganization decision; §3.2 describes the single network that serves both.

Stage 1: attach navigation. Placing an item anywhere in the tree is a choice among $O(|T|)$ locations; we model it instead as greedy navigation over local candidates. At the current node v (initially the root), the candidate set is $\mathcal{N}(v) = \{v\} \cup \{\text{internal children of } v\}$—"stop here" plus one option per child category, since items attach only under internal nodes. The policy selects ATTACH (terminating) or DESCEND into one of the children. Descent ends the first time ATTACH is chosen, or is forced when v has no internal children.

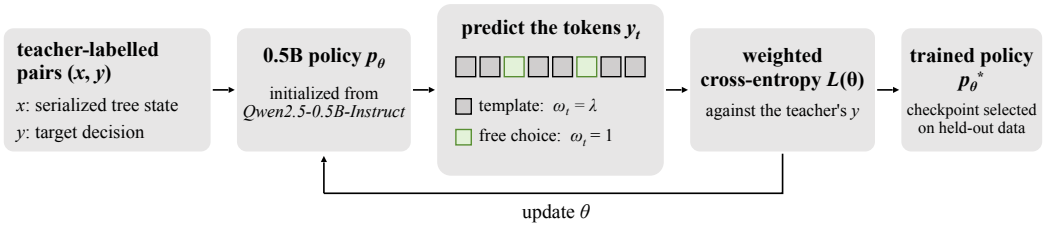

Figure 3: How the policy is trained. The teacher's labelled decisions are pairs of a serialized state x and the answer y that names the decision. The 0.5B policy, initialized from Qwen2.5-0.5B-Instruct, reads x and predicts the tokens of y; the weighted cross-entropy of Eq. 1 compares them with the teacher's answer, weighting the free choice points fully and the template tokens by λ, and the update is applied to θ. The checkpoint selected on held-out data is the policy used to build every tree.

Stage 2: the closure. We now give the closure precisely. Around the focal item c it is the three-level window rooted at c's grandparent g. It consists of c's parent p and three sets of nodes beneath them. The *operable set* O, of size K, holds c and its sibling items; these are the items the policy edits. The *demotion-target set* S, of size M, holds c's sibling internal nodes; these are the only legal DEMOTE destinations. The *read-only context* R holds the other children of g: it can neither be edited nor targeted, but it lets the policy see how the neighborhood is organized one level up. The teacher and the student observe the same closure.

Action space. Every operable item $i \in O$ receives exactly one edit a_i, drawn from the four types IDLE, PROMOTE, DEMOTE and GROUP. A DEMOTE additionally carries a target $\tau_i \in \{0, \ldots, M-1\}$ naming which element of S the item sinks into; the other three take no argument. IDLE stays; PROMOTE lifts the item to the grandparent (illegal when p is the root); DEMOTE sinks it into the sibling subtree $S[\tau_i]$ (a cycle guard applies); GROUP founds new structure—all GROUP-marked items of the round move into *one* newly created internal node under p, whose category name an LLM writes from the grouped items (*sequential single-group* semantics: at most one new group per pass, so GROUP carries no learnable membership parameters). A category is never founded for a single item (singleton downgrade to IDLE); the teacher additionally requires a group of at least three and at most $\max(3, \lceil 0.6K \rceil)$ items (§3.3).

3.2 The Policy

Both decisions are made by a single **0.5B policy network**, initialized from Qwen2.5-0.5B-Instruct. It reads the state as text—the incoming item and the legal candidates $\mathcal{N}(v)$ for Stage 1, the closure C for Stage 2—and writes the decision as text in a fixed grammar. One network serves both stages, since the two grammars diverge at the second token it emits; no task label is supplied at input or used by the loss.

A malformed or illegal edit would corrupt the structure, so the policy does not emit free text. We decode against the grammar of the answer: at each step a state machine admits only the tokens that can continue a well-formed, legal one, and masks the rest. The grammar leaves three free choices per operable item—whether it joins a GROUP, which of the remaining types it takes, and, for a DEMOTE, which element of S—and one for Stage 1. Every other position is forced. Schema validity and legality therefore hold by construction rather than by training.

3.3 Learning from a Teacher

Calling a large LLM for every decision is what makes agentic ingestion expensive, so we distill the decisions of a frozen teacher LLM into the policy (Figure 3).

Teacher labeling. We run the round of §3.1 over a training corpus with the teacher making both decisions. The corpus is cut into buckets, each a contiguous body of text such as one book, one hearing or one filing, and each bucket plays the role of an entity e whose segments arrive as $\mathcal{F}_e$. At

every decision point the teacher sees the same serialized state x the policy will see, and its answer y is the label. Answers that fail validation are retried or left out of the training data (Appendix B). To measure label quality, each of 500 held-out decisions is answered twice by an independent frontier judge, gpt-6-sol, which sees no other answer and lists the other answers it would also accept; an answer is counted correct when it is in this accepted set, since many decisions admit more than one reasonable outcome. The teacher is right on 80.6% of them (85.0% of attach and 77.7% of reorganization decisions), against 86.4% for gpt-6-luna, a frontier reference (Table 4 in Appendix B).

Every validated decision becomes one training pair (x, y), and the pairs are split into training and validation by whole tree.

Objective. The student is trained on the teacher's answers with a token-level cross-entropy (Kim & Rush, 2016), as in standard supervised finetuning, with one change. Most answer tokens are template—punctuation and key names that the grammar forces at decoding time anyway—so we down-weight them and keep full weight on the free choice points, the positions the grammar leaves open (the group gate, the action value, the demotion target and the candidate index). For a batch $\mathcal{B}$ of pairs, with y_t the t-th answer token,

$$\mathcal{L}(\theta) = -\sum_{(x,y)\in\mathcal{B}} \sum_t w_t \log p_\theta(y_t \mid x, y_{<t}) \,/\, \sum_{(x,y)\in\mathcal{B}} \sum_t w_t, \tag{1}$$

where $w_t = 1$ at a free choice point and $w_t = \lambda = 0.1$ elsewhere; $\lambda = 1$ recovers the standard loss. Once trained, the policy replaces the teacher for every decision, and an LLM is called only to name the categories that GROUP founds.

3.4 RETRIEVAL

A query is answered from the facts visible to it: on a growing memory, the tree snapshot at the time the query is asked. A floor score combines bge-m3 cosine similarity with BM25. The floor's top 200 are then reordered by three fixed tree factors, which read the hierarchy the policy has built—the continuation of a top-ranked fact in its own leaf aspect, the facts filed with it around the same time or under a well-named aspect, and the newest facts of the aspects the top-ranked facts belong to—and the resulting top 40 are rescored pointwise by a reranker that sees only the query and the candidate text. Appendix G specifies every step and weight.

4 EXPERIMENTS

We evaluate the memory the policy builds by the only thing that ultimately matters about a memory: what a query can retrieve from it. Every method under comparison receives the same source units and the same queries, and its top ten results are scored by the same scorer, so that the only difference between two columns is how the memory was organized and searched.

4.1 SETUP

Datasets. We use three public datasets as benchmarks: two are evaluated online and one offline, in the sense defined below. **CloneMem** (Hu et al., 2026) evaluates long-term memory for AI clones. Each of its ten personas leaves one to three years of non-conversational digital traces, such as diary entries, social-media posts and emails, 8,489 items in all, and its 1,187 questions each carry the time at which they are asked. **RealMem** (Bian et al., 2026) consists of long-term, project-oriented conversations between ten users and an assistant, 2,055 dialogue sessions in all, with 1,415 questions that arise inside the sessions. **NovelQA** (Wang et al., 2025a), as packaged for retrieval by LMEB (Zhao et al., 2026), has 1,541 questions over 79,286 passages from 61 novels. We score every benchmark by recall of its gold evidence in the top ten retrieved items (R@10), with no answer generation or LLM judge, and follow each benchmark paper's own definition so that our numbers are comparable with the published ones. For a query q with gold items G_q and top ten retrieved items T_q, CloneMem and NovelQA use *fractional* recall and RealMem uses *all-gold* recall, $r_q^{\text{frac}} = |G_q \cap T_q|/|G_q|$ and $r_q^{\text{all}} = \mathbf{1}[G_q \subseteq T_q]$: the share of the gold items found in the top ten, and whether all of them are. Scores are averaged per persona on CloneMem and RealMem and over queries on NovelQA (Appendix D), so absolute values are comparable only between methods on the same benchmark.

Online and offline evaluation. Under *online* evaluation, questions are asked while the memory is still being ingested: each question is answered from the memory built so far, which holds only the items that arrived before it. Under *offline* evaluation, all items are ingested first and every question is then answered from the complete memory. CloneMem and RealMem are evaluated online, following the time order of their items; NovelQA is evaluated offline, one book at a time.

Baselines. We compare against four open-source memory systems, each run on its own native configuration and at the model tier it was published on: **Mem0** (Chhikara et al., 2025), **A-Mem** (Xu et al., 2025), **MemOS** (Li et al., 2025) and **HippoRAG 2** (Gutiérrez et al., 2025). We additionally report a **Flat** baseline, which is cosine-similarity retrieval over embeddings: every source unit is embedded whole with bge-m3 (8,192-token window, no query instruction), and the units are ranked by the cosine similarity between their embedding and the query's. There is no structure, no lexical channel, no reranking and no LLM call, so Flat isolates what organization contributes over none. We reproduce each of the four memory systems faithfully, with the configuration its authors released. Their native retrieval components differ: A-Mem ranks its notes by embedding cosine alone; Mem0 re-weights its cosine candidates with BM25 and an entity score; MemOS reranks its cosine candidates with a cross-encoder (bge-reranker-v2-m3); and HippoRAG 2 filters retrieved facts with an LLM call and ranks passages by Personalized PageRank over its graph.

Training data. The policy is trained on text from 33 public Hugging Face datasets, many of them book collections such as Project Gutenberg,[1] together with dialogue, textbooks, and technical, legal, academic and corporate documents (Appendix A).

Policy training. The labels come from the frozen Qwen3.8-27B teacher of §3.3. The student, Qwen2.5-0.5B-Instruct, is fully finetuned as a single model for both decisions on 35.3 million labeled examples, and we keep the checkpoint whose reorganization actions agree best with the teacher's on held-out closures, measured by Cohen's κ (Appendix A). When a tree is built over a benchmark, every routing and reorganization decision is made by this policy, and the category that a GROUP founds is named by gpt-4o-mini, the only LLM API call our method makes during ingestion.

Is the policy trained well? We measure the policy exactly as the teacher is measured in §3.3, on the same 500 held-out decisions against the same judge. The trained policy is right on 78.8% of them, 97.8% of the teacher's 80.6%. Training is what closes the gap: a random legal action is right on 21.7%, and the untrained base model, Qwen2.5-0.5B-Instruct, is right on only 15.9% of the attach decisions for which it produces a legal answer, below the 23.9% of a random legal attach, and produces no legal reorganization answer at all.

Cost accounting. *Tokens* count the input and output tokens of every model call a method makes during ingestion and retrieval over the full benchmark; embedding and reranking calls are not counted for any method, and for ours neither are the calls of the 0.5B policy, so what remains are the gpt-4o-mini calls that name new categories. *Time* in Table 1 is measured on three buckets of each benchmark (186, 442 and 65 queries), serially with one request in flight, as the time spent in a method's ingestion and indexing calls (*build*) and in its search calls (*retrieve*); Appendix D gives the hardware and further details.

Calibration against published numbers. Where a published number exists for the same method and benchmark (grey in Table 1), our reproduction is within three points in five of six cells; Mem0 on CloneMem is seven points higher because we run its current release, whose hybrid retrieval adds BM25 and an entity score.

4.2 RESULTS

Table 1 gives the main comparison. Our system reaches R@10 of 35.70, 71.46 and 60.67 on CloneMem, RealMem and NovelQA, comparable to the strongest comparator, HippoRAG 2, at 32.76, 67.86 and 57.56. The tree and the search that reads it (Appendix G) bring a small gain. This is to

[1] https://huggingface.co/datasets/sedthh/gutenberg_english

Table 1: Retrieval quality and cost on the three benchmarks. Grey numbers in parentheses are the published R@10 of the same method on the same benchmark (Hu et al., 2026; Bian et al., 2026; Zhao et al., 2026). Tokens and time are defined in §4.1, *Cost accounting*.

Dataset	Method	R@10	Tokens in/out	Build/Retrieve
CloneMem (1,187 queries)				
	Flat	27.96 (27.52)	0 / 0	3.70 s / 0.20 s
	Mem0	24.61 (17.54)	85.09M / 3.23M	9,918 s / 271 s
	A-Mem	25.71 (26.87)	41.41M / 5.27M	11,951 s / 127 s
	MemOS	23.61	16.25M / 7.37M	13,143 s / 540 s
	HippoRAG 2	32.76	16.72M / 2.71M	5,839 s / 353 s
	Ours	**35.70**	1.40M / 0.0053M	558 s / 258 s
RealMem (1,415 queries)				
	Flat	63.74	0 / 0	16.32 s / 0.76 s
	Mem0	60.74 (60.55)	21.94M / 0.62M	4,543 s / 859 s
	A-Mem	60.43 (57.60)	39.92M / 9.21M	47,797 s / 12 s
	MemOS	58.62	26.42M / 3.40M	14,902 s / 356 s
	HippoRAG 2	67.86	11.65M / 1.01M	6,891 s / 875 s
	Ours	**71.46**	1.10M / 0.0016M	554 s / 1,537 s
NovelQA (1,541 queries)				
	Flat	44.93 (42.63)	0 / 0	2.44 s / 0.11 s
	Mem0	39.09	769.64M / 13.74M	8,432 s / 83 s
	A-Mem	29.40	234.19M / 79.01M	19,364 s / 0.79 s
	MemOS	52.88	540.21M / 164.20M	9,411 s / 187 s
	HippoRAG 2	57.56	85.94M / 13.71M	4,645 s / 110 s
	Ours	**60.67**	7.41M / 0.0534M	538 s / 64 s

be expected: ingestion spends very few tokens and adds little information beyond the items themselves, which are kept verbatim, and the names of the categories that group them. On top of cosine similarity, BM25 and the reranker, the tree factors add $+0.92$, $+0.25$ and $+0.18$ points, and the increment is statistically clear on CloneMem. The comparators differ from each other in their retrieval components as well: A-Mem ranks by cosine similarity alone, Mem0 adds BM25 and an entity score, MemOS reranks with a cross-encoder, and HippoRAG 2 filters facts with an LLM and ranks passages by Personalized PageRank (§4.1, *Baselines*). Where our system differs clearly is the cost of building the memory. On the three timed buckets of each benchmark it builds in 558, 554 and 538 s of serial wall-clock time, against 4,543 to 47,797 s for the four memory systems, and over the full benchmarks its only hosted-model calls, which name the aspects, take 1.10M to 7.41M input tokens, against 11.65M to 769.64M.

4.3 Ablation Study

Beyond BM25 and the reranker, the components we ablate are the tree factors of our search (Appendix G). Each targets one class of question: F1 lookup (C1), F2 multi-evidence (C2) and F3 latest state (C3) (Table 7). A factor reads only the tree, never the dataset or the question type, so it carries over to any benchmark with questions of its class. The factors were designed on LoCoMo, a separate development set, where they add $+0.93$ on top of BM25 and the reranker, and were then applied unchanged to the three benchmarks (Appendix G). BM25 and the reranker add $+6.82/+7.47/+15.56$ on CloneMem/RealMem/NovelQA, and the tree factors a further $+0.92/+0.25/+0.18$. The caption of Table 2 explains how to read it.

4.4 Limitations

Our factors come from ideas we supplied, which an LLM coding agent turned into closed-form scores, and each implements one way of locating evidence in the tree (Appendix G). They therefore help only questions whose evidence one of these three heuristics can reach; evidence related in other ways, for instance across branches that share neither a name nor a time window, is left to cosine similarity, BM25 and the reranker, and other kinds of questions would need other factors. Their gains are small, although together the three improve on cosine similarity alone on every benchmark

Table 2: Ablation of our retrieval (R@10, %). **How to read:** Base is cosine similarity alone; each following row adds the listed components to Base, and Ours adds F1–F3, BM25 and the reranker; the factors are named in Table 7. Column pairs give R@10 and the gain Δ over Base on one question class (Table 7); *All* is the whole benchmark. Bold: 95% paired-bootstrap CI excludes 0; NA: not applicable.

	C1 lookup		C2 multi-evidence		C3 latest state		All	
Method	R@10	Δ	R@10	Δ	R@10	Δ	R@10	Δ
(a) CloneMem								
Base (cosine)	36.99	0.00	24.46	0.00	38.72	0.00	27.93	0.00
+ F1	40.19	**+3.20**	25.33	+0.87	38.68	−0.04	28.83	+0.90
+ F2	38.67	+1.68	25.01	**+0.54**	37.76	−0.95	28.38	+0.45
+ F3	36.92	−0.07	25.26	**+0.80**	42.86	**+4.15**	29.20	**+1.27**
+ F1 + F2 + F3	41.91	**+4.92**	26.41	**+1.95**	41.66	**+2.94**	30.28	**+2.35**
Ours	43.14	+6.15	32.47	**+8.01**	45.67	**+6.96**	35.70	**+7.78**
(b) RealMem								
Base (cosine)	51.67	0.00	NA	NA	63.65	0.00	63.35	0.00
+ F1	51.67	+0.00	NA	NA	63.28	−0.37	63.00	−0.36
+ F2	51.67	+0.00	NA	NA	64.02	+0.37	63.72	+0.36
+ F3	56.67	+5.00	NA	NA	65.73	**+2.08**	65.51	**+2.16**
+ F1 + F2 + F3	56.67	+5.00	NA	NA	65.24	**+1.59**	65.04	**+1.68**
Ours	60.00	+8.33	NA	NA	71.74	**+8.09**	71.46	**+8.11**
(c) NovelQA								
Base (cosine)	50.03	0.00	28.97	0.00	NA	NA	45.00	0.00
+ F1	50.47	+0.44	28.64	−0.33	NA	NA	45.26	+0.26
+ F2	50.49	+0.46	29.78	+0.81	NA	NA	45.55	+0.55
+ F3	50.03	−0.00	28.87	−0.10	NA	NA	44.97	−0.02
+ F1 + F2 + F3	50.57	+0.54	27.86	−1.11	NA	NA	45.15	+0.15
Ours	63.97	**+13.94**	50.16	**+21.19**	NA	NA	60.67	**+15.67**

(Table 2), and they depend on the trees having structure to read. The factors were designed on LoCoMo, where they add $+0.93$; on LongMemEval they add almost nothing (Table 8), because the trees our policy builds there are nearly flat and cosine similarity alone is already close to saturation. Several classes are also small, so their cells are noisy.

5 Conclusion

We set out to grow a structured memory online, one item at a time, without the per-item LLM calls that make agentic ingestion expensive. First, ingestion can be cast as local maintenance: each decision reads only a small window of the tree that does not grow with it (Table 6). Second, a 0.5B policy is enough to make these decisions: it is right on 78.8% of held-out decisions, 97.8% of its teacher, and during ingestion an LLM is called only to name the categories the policy founds. Third, retrieval can draw on the hierarchy beyond flat similarity: the tree factors add a small but measurable gain on top of BM25 and the reranker, clearest under online arrival. Together, our system reaches retrieval quality comparable to that of state-of-the-art memory systems, with at least ten times fewer hosted-LLM tokens and at least eight times less build time than any of them.

The factors are closed-form scores that an LLM agent derived from ideas we supplied, so they help only the kinds of questions they were designed for. Learning such factors from data is a natural next step, as is carrying the recipe beyond trees, to any structure whose maintenance can be cut into decisions over a bounded region, such as a graph or a vector store with explicit groups.

AI Use Statement

Large language models are part of the method itself and are described where they are used: a frozen teacher labels the training decisions (§3.3), gpt-4o-mini names new categories, a reranker scores candidates at retrieval, and gpt-6-sol and gpt-6-luna judge label quality (Appendix B); every prompt

is given in Appendix H. Beyond these roles, the tree factors were implemented as closed-form scores by an LLM coding agent from ideas the authors supplied (Appendix G). LLM assistants were also used to propose and design auxiliary experiments and audits, to implement, debug and run experiment code, to interpret results, and to draft and edit the text. The authors decided the research questions and the method, reviewed every experiment, verified every number and claim in the paper, and take full responsibility for its content.

REFERENCES

Haonan Bian, Zhiyuan Yao, Sen Hu, Zishan Xu, Shaolei Zhang, Yifu Guo, Ziliang Yang, Xueran Han, Huacan Wang, and Ronghao Chen. RealMem: Benchmarking LLMs in real-world memory-driven interaction. *arXiv preprint arXiv:2601.06966*, 2026.

Han Chen, Zining Zhang, Wenqi Pei, Bingsheng He, Ming Wu, Jason Zeng, Michael Heinrich, Wei Wu, and Hongbao Zhang. MemForest: An efficient agent memory system with hierarchical temporal indexing. *arXiv preprint arXiv:2605.23986*, 2026.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready AI agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.

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

Jizhan Fang, Xinle Deng, Haoming Xu, Ziyan Jiang, Yuqi Tang, Ziwen Xu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, Huajun Chen, and Ningyu Zhang. LightMem: Lightweight and efficient memory-augmented generation. *arXiv preprint arXiv:2510.18866*, 2025.

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

Bernal Jiménez Gutiérrez, Yiheng Shu, Weijian Qi, Sizhe Zhou, and Yu Su. From RAG to memory: Non-parametric continual learning for large language models. *arXiv preprint arXiv:2502.14802*, 2025.

Sen Hu, Zhiyu Zhang, Yuxiang Wei, Xueran Han, Zhenheng Tang, Huacan Wang, and Ronghao Chen. CloneMem: Benchmarking long-term memory for AI clones. *arXiv preprint arXiv:2601.07023*, 2026.

Minki Kang, Wei-Ning Chen, Dongge Han, Huseyin A. Inan, Lukas Wutschitz, Yanzhi Chen, Robert Sim, and Saravan Rajmohan. ACON: Optimizing context compression for long-horizon LLM agents. *arXiv preprint arXiv:2510.00615*, 2025.

Yoon Kim and Alexander M. Rush. Sequence-level knowledge distillation. In *Conference on Empirical Methods in Natural Language Processing*, 2016.

Zhiyu Li, Chenyang Xi, Chunyu Li, Ding Chen, Boyu Chen, Shichao Song, Simin Niu, Hanyu Wang, Jiawei Yang, Chen Tang, Qingchen Yu, Jihao Zhao, Yezhaohui Wang, Peng Liu, Zehao Lin, Pengyuan Wang, Jiahao Huo, Tianyi Chen, Kai Chen, Kehang Li, Zhen Tao, Huayi Lai, Hao Wu, Bo Tang, Zhengren Wang, Zhaoxin Fan, Ningyu Zhang, Linfeng Zhang, Junchi Yan, Mingchuan Yang, Tong Xu, Wei Xu, Huajun Chen, Haofen Wang, Hongkang Yang, Wentao Zhang, Zhi-Qin John Xu, Siheng Chen, and Feiyu Xiong. MemOS: A memory OS for AI system. *arXiv preprint arXiv:2507.03724*, 2025.

Jun Liu, Zhenglun Kong, Peiyan Dong, Changdi Yang, Tianqi Li, Hao Tang, Geng Yuan, Wei Niu, Wenbin Zhang, Pu Zhao, Xue Lin, Dong Huang, and Yanzhi Wang. Structured agent distillation for large language model. In *International Conference on Autonomous Agents and Multiagent Systems*, 2026a.

Yifan Simon Liu, Liam Gallagher, Faeze Moradi Kalarde, Jiazhou Liang, Armin Toroghi, and Scott Sanner. Temporal order matters for agentic memory: Segment trees for long-horizon agents. *arXiv preprint arXiv:2606.04555*, 2026b.

Yikang Liu, Yeting Shen, Hongao Zhu, Lilong Xu, Zhiheng Qian, Siyuan Song, Kejia Zhang, Jialong Tang, Pei Zhang, Baosong Yang, Rui Wang, and Hai Hu. A systematic assessment of language models with linguistic minimal pairs in Chinese. *Transactions of the Association for Computational Linguistics*, 14:755–771, 2026c. doi: 10.1162/tacl.a.648.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. *arXiv preprint arXiv:2304.03442*, 2023.

Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. Zep: A temporal knowledge graph architecture for agent memory. *arXiv preprint arXiv:2501.13956*, 2025.

Alireza Rezazadeh, Zichao Li, Wei Wei, and Yujia Bao. From isolated conversations to hierarchical schemas: Dynamic tree memory representation for LLMs. *arXiv preprint arXiv:2410.14052*, 2024.

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

Haoran Sun and Shaoning Zeng. Hierarchical memory for high-efficiency long-term reasoning in LLM agents. *arXiv preprint arXiv:2507.22925*, 2025.

Cunxiang Wang, Ruoxi Ning, Boqi Pan, Tonghui Wu, Qipeng Guo, Cheng Deng, Guangsheng Bao, Xiangkun Hu, Zheng Zhang, Qian Wang, and Yue Zhang. NovelQA: Benchmarking question answering on documents exceeding 200k tokens. In *International Conference on Learning Representations*, 2025a.

Yu Wang, Ryuichi Takanobu, Zhiqi Liang, Yuzhen Mao, Yuanzhe Hu, Julian McAuley, and Xiaojian Wu. Mem-α: Learning memory construction via reinforcement learning. *arXiv preprint arXiv:2509.25911*, 2025b.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-MEM: Agentic memory for LLM agents. *arXiv preprint arXiv:2502.12110*, 2025.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Jinhe Bi, Kristian Kersting, Jeff Z. Pan, Hinrich Schütze, Volker Tresp, and Yunpu Ma. Memory-R1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. *arXiv preprint arXiv:2508.19828*, 2025.

Yi Yu, Liuyi Yao, Yuexiang Xie, Qingquan Tan, Jiaqi Feng, Yaliang Li, and Libing Wu. Agentic memory: Learning unified long-term and short-term memory management for large language model agents. *arXiv preprint arXiv:2601.01885*, 2026.

Guibin Zhang, Hejia Geng, Xiaohang Yu, Zhenfei Yin, Zaibin Zhang, Zelin Tan, Heng Zhou, Zhongzhi Li, Xiangyuan Xue, Yijiang Li, Yifan Zhou, Yang Chen, Chen Zhang, Yutao Fan, Zihu Wang, Songtao Huang, Francisco Piedrahita-Velez, Yue Liao, Hongru Wang, Mengyue Yang, Heng Ji, Jun Wang, Shuicheng Yan, Philip Torr, and Lei Bai. The landscape of agentic reinforcement learning for LLMs: A survey. *arXiv preprint arXiv:2509.02547*, 2025.

Jiaquan Zhang, Chaoning Zhang, Shuxu Chen, Zhenzhen Huang, Pengcheng Zheng, Zhicheng Wang, Ping Guo, Fan Mo, Sung-Ho Bae, Jie Zou, Jiwei Wei, and Yang Yang. Lightweight LLM agent memory with small language models. In *Annual Meeting of the Association for Computational Linguistics*, 2026.

Xinping Zhao, Xinshuo Hu, Jiaxin Xu, Danyu Tang, Xin Zhang, Mengjia Zhou, Yan Zhong, Yao Zhou, Zifei Shan, Meishan Zhang, Baotian Hu, and Min Zhang. LMEB: Long-horizon memory embedding benchmark. *arXiv preprint arXiv:2603.12572*, 2026.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. MemoryBank: Enhancing large language models with long-term memory. *arXiv preprint arXiv:2305.10250*, 2023.

A Policy Training Details

This section gives the details of the training pipeline of §3.3 that the setup of §4.1 leaves out. Table 3 collects the settings.

Table 3: Training and inference settings of the policy.

Teacher labeling	
Teacher	Qwen3.8-27B, frozen, thinking mode
Serving	vLLM, bf16 weights, fp8 KV cache
Labeled trees	38,764 trees of 300 or 700 items, 33 public sources
Supervised finetuning	
Initialization	Qwen2.5-0.5B-Instruct, full finetuning, one model for both decisions
Training examples	35,335,221 (21,845,272 attach, 13,489,949 reorganization)
Training tokens	18.8B; longest example 29,141 tokens
Validation examples	60,051, held out by whole tree
Loss	Eq. 1, answer tokens only, $\lambda = 0.1$
Optimizer	AdamW, no weight decay
Learning rate	1×10^{-5}, constant after warmup over the first 1% of steps
Precision	bf16 mixed precision
Batching	token budget, 16,384 tokens per GPU
Hardware	up to 40 NVIDIA B200 GPUs, data parallel
Training length	about 1.45 epochs
Checkpoint selection	highest Cohen's κ on held-out reorganization decisions
Inference when building trees	
Serving	vLLM, greedy decoding, context 32,768 tokens
Category naming	gpt-4o-mini, one call per GROUP

Sources. The labeled trees are drawn from 33 publicly available Hugging Face datasets (§4.1, *Training data*). Each is pinned to a specific commit, so the corpus can be rebuilt exactly.

From raw text to items. No LLM is involved in preparing the training text: the policy is trained on the source text itself. Each document is first cleaned—bibliographies, markup, citations, page numbers, tables, figure captions and repeated running heads are removed—and paragraphs that are not prose are dropped, such as tables of contents, headings, captions, lines dominated by capitals, digits or symbols, and fragments too short to end a sentence. The remaining paragraphs are split into sentences, and consecutive sentences are packed greedily into items of about 450 characters, never exceeding 900 and carried across a paragraph break only while shorter than 240; an overlong sentence is cut at punctuation. Items average about 490 characters. Dialogue sources are handled the same way at the level of turns: consecutive turns are packed into items of the same length, with each speaker's name kept in the text, but never across a section boundary such as a new scene or debate item.

Buckets. The items of one coherent document form a *bucket*, kept in reading order. What counts as a document depends on the source: one book, one patent, one company filing, one criminal trial, one day of parliamentary or congressional record, one chat channel on one day, or the whole journal of one hiker. Documents shorter than 2,000 characters and buckets with fewer than nine items are discarded. Because several book collections overlap, books are deduplicated across sources by their identifiers, their titles and a hash of their opening text, which removes 89,327 buckets.

Sampling training trees. The training pool consists of the 264,478 buckets with at least 500 items and a unique identifier. The buckets are split into 220,000 for training and 44,478 for validation by a hash of their identifier. From each bucket we draw two independent uniform samples of 300 items, each seeded by the bucket identifier, and each sample becomes one tree whose items are presented to the labeler in shuffled order, so that the order of arrival carries no information about where an item belongs. Both trees of a bucket stay on the same side of the split. We label the trees of 30 training shards of this pool, together with trees of 700 items drawn once each from training buckets with at least 700 items outside those shards, and train on 38,764 labeled trees.

Held-out split. Trees, not records, are assigned to the training or validation side, and the training side then drops every record whose prompt also occurs in validation; this removes 290,130 records, 0.8% of the pool.

Overlap with NovelQA. The novels of NovelQA are largely public-domain books, and several of them are also in the book collections we train on. To avoid data leakage, we ensured that no NovelQA book enters our training or validation set. Although a number of NovelQA books are indeed present in our book collections, we deliberately excluded them when constructing the training and validation data, so that no book-level overlap exists between the two. As a result, training never encounters any NovelQA content, and no NovelQA question, gold passage or retrieval signal is ever seen in training.

Template share of the answer. For the attach decision 87.9% of the answer tokens are template, leaving roughly one token per example that carries the decision; for reorganization the figure is 79.0%. Training unweighted would therefore spend most of the gradient on punctuation, which is what the weights of Eq. 1 prevent.

Checkpoint selection. Checkpoints are written every half epoch and each is decoded, under the output grammar, on 3,000 held-out closures (18,104 reorganization decisions). We compare its actions with the teacher's by Cohen's κ,

$$\kappa = \frac{p_o - p_e}{1 - p_e},$$

where p_o is the fraction of decisions on which the two agree and p_e is the agreement expected by chance from their marginal action frequencies. Raw agreement would be misleading here: IDLE accounts for most labels, so a policy that never moves anything already agrees with the teacher on most decisions, while its κ is zero. The checkpoint with the highest κ is kept.

B TEACHER LABELING DETAILS

Procedure. Each training tree is labeled by running the round of §3.1 with the teacher in place of the policy, and every teacher answer is applied, so the tree grows as the teacher would grow it. The tree starts as the root alone and the items arrive one at a time. For the attach decision the teacher is asked at every step of the descent: it sees the incoming item and the current node together with its child categories, and answers either to attach at the current node or to descend into one child, until it attaches. For the reorganization decision the closure around the item that has just been attached is serialized and the teacher assigns an edit to every operable item; the edits are applied and the step is repeated on the same focal item, stopping when the edits change nothing, when the focal item itself moves, or when a GROUP founds a new category. On complete trees this amounts to 2.43 attach decisions and 1.30 reorganization decisions per item. Every decision is recorded as a pair of the serialized state and the teacher's answer.

Prompts. Both decisions use fixed prompts that were not changed during the run. The attach prompt has a 260-token system message asking for the most specific parent for the item, and a user message listing the candidates by number; the teacher replies with a short reason and the chosen index. The reorganization prompt has a 2,103-token system message that defines the tree, the closure and the four edits, gives one worked example and a legality checklist, and asks for one JSON line per operable item; the user message is the closure serialized as JSON, including which edits are

legal in it. Stage-2 answers are produced in thinking mode and Stage-1 answers without it; only the final answer is parsed. Decoding is greedy.

Validation. A reorganization answer is accepted only if it assigns exactly one edit to every operable item, every DEMOTE names an existing child category, every PROMOTE is legal at that depth, every GROUP has at least three members, and a new category is not named after its parent or the root. A GROUP larger than $\max(3, \lceil 0.6K \rceil)$ members is trimmed to that size before validation, the members left out becoming IDLE. An illegal answer is sent back with the error for up to two further attempts, and a timed-out call is retried up to three times. When the teacher still fails, the round is applied as all IDLE so that the tree can continue, and that decision is excluded from the training data; this happened on 0.048% of reorganization calls in the data we train on. An attach answer that cannot be parsed after retries falls back to attaching at the current node.

Labels. A labeling run is accepted when the teacher gives up on fewer than 0.5% of reorganization calls and IDLE stays at or below 90% of decisions. The accepted labels hold 5.96 items per closure on average (median 5, maximum 203) and leave 39.4% of closures unchanged. Over all reorganization decisions the actions split into IDLE 72.46%, DEMOTE 11.70%, PROMOTE 8.49% and GROUP 7.35%.

Table 4: Share of 500 held-out decisions answered correctly, judged by gpt-6-sol.

Answerer	Attach (200)	Reorganization (300)	All (500)
gpt-6-luna (frontier reference)	94.0	81.3	86.4
Teacher (Qwen3.8-27B)	85.0	77.7	80.6

Label quality. §3.3 and §4.1 measure the teacher and the policy on 500 held-out decisions, 200 attach and 300 reorganization. gpt-6-sol answers each decision twice without seeing any other answer, and after each answer lists the other answers it would also accept. The set of accepted answers is the union of its two answers and the alternatives it lists. For 340 decisions the judge gives the same answer twice and accepts no alternative, so the accepted set is that single answer; for the other 160 it holds several answers, reflecting that many decisions have more than one reasonable outcome. An answer is scored correct when it is in the accepted set. On the 340 single-answer decisions the teacher is right on 77.4% and the policy on 72.9%; on the 160 others, on 87.5% and 91.3%. For random legal actions we report the exact expectation, the share of legal actions that are in the accepted set. gpt-6-luna answers the same decisions with the judge's prompt; since it comes from the same model family as the judge and sees the rules as the judge states them, its 86.4% is a reference point rather than a strict upper bound.

As an independent check of a different kind, the same judge also rates teacher decisions directly. Shown one teacher decision at a time on held-out reorganization closures, it rates each as clearly right, defensible or wrong. Of 200 decisions drawn from the natural label distribution, 86.5% are rated right or defensible (95% CI $[81.8, 91.4]$, resampling closures). The judge is not merely lenient: in the same blind pool it accepts only 39.0% of random legal actions, and it gives the same verdict to 97.2% of repeated items.

Scale and cost. The teacher is served by vLLM 0.27.1 with bf16 weights and an fp8 KV cache, one instance per NVIDIA B200 GPU and up to 200 concurrent sequences per instance. The labeling run lasted 10.5 days on between 10 and 46 such instances, about 26 on average, for about 6,500 GPU-hours in total. The run labeled 23.2 million attach and 14.3 million reorganization decisions over 38,764 trees, counting the validation side; after the held-out split, the training store keeps the 35.3 million examples of Table 3. The teacher read at least 3.8×10^{10} prompt tokens and generated at least 5.7×10^{10} tokens including its thinking to produce them; these are lower bounds, since part of the serving logs was lost on restarts. Appendix C describes the system that ran the labeling.

C Labeling System

Labeling a tree is inherently sequential: every decision reads the tree left by the one before it, and an item cannot be routed until the previous item's reorganization has been applied. The labeling system

therefore draws its parallelism from running many trees at once, each advancing one decision at a time, and is organized so that the thousands of trees in flight keep every teacher instance saturated. Table 5 summarizes it.

Clients: processes and threads. A coordinator splits its trees round-robin into one static shard per worker process and starts N CPU-only worker processes, each running $M = 25$ driver threads. A driver takes the next tree from its process's shard and grows it from start to finish with the real tree code of §3.1, the teacher standing in for the policy: it builds each prompt, sends a blocking request, validates the answer, applies the chosen action and moves on to the next decision. Because a blocked request releases the interpreter lock, a process with M drivers keeps M requests in flight, and a coordinator keeps about $N \times M$. We ran up to four coordinators at once with N between 40 and 200, and giving each process its own shard, instead of a shared queue, removed the lock contention that otherwise dominated worker time.

Servers and routing. The teacher runs as one vLLM instance per GPU, without tensor parallelism, with prefix caching, chunked prefill and up to 200 concurrent sequences; reorganization requests enable the teacher's thinking mode and attach requests do not. Requests are routed on the client side: every call takes the next entry of a shared endpoint list, and a balancer reads the running and waiting requests of each instance every 10 s and rewrites the list so that less loaded instances appear more often. An instance joins the list only after answering a real request, and can be removed without restarting any client, which let the pool grow and shrink between 10 and 46 instances during the run. Across the run the pool held a median of about 4,800 requests in flight (90th percentile about 7,800), about 200 per instance, the concurrency at which a single instance reached its highest throughput in our tests.

Streaming and fault tolerance. Decisions are written as they are made: each tree appends its records to its own file, which is synced and renamed only when the tree is complete, and a completion marker is then written. A restarted coordinator skips completed trees and regrows unfinished ones, replaying their earlier decisions from a content-addressed cache of successful teacher answers instead of querying the teacher again. Workers exit if their coordinator disappears, so no orphaned client keeps loading the servers. A request is retried up to three times on a timeout or network error; an answer that fails validation is retried up to three times with the validation errors and the previous answer appended to the prompt; and a decision the teacher still gives up on is recorded as a default (attach at the root, or keep every item in place), flagged, and excluded from training while the tree continues.

Table 5: The labeling system.

Unit of parallelism	one tree per driver thread; decisions within a tree are sequential
Clients	up to 4 coordinators, each $N = 40$–200 processes $\times$ $M = 25$ threads
Servers	one vLLM instance per B200 GPU, 10–46 instances, ≤ 200 sequences each
Routing	client-side, load-weighted endpoint list refreshed every 10 s
Requests in flight	median $\approx$ 4,800, 90th percentile $\approx$ 7,800
Decoding	temperature 0; thinking on for reorganization, off for attach
Output	per-tree streaming files, completion markers, answer cache

D Retrieval Metrics

Every R@10 in Table 1 is computed from the ranked list a method returns and the gold annotations of the benchmark; no answer is generated and no LLM is involved in scoring. This section fixes the notation used in §4.1 and states, for each benchmark, which rule its official scorer applies and how we checked it.

Notation. Let $\mathcal{Q}$ be the queries of a benchmark. For a query $q \in \mathcal{Q}$, G_q is its set of gold source identifiers: the original items that contain the evidence for q. A method answers q with a ranked list of source identifiers; a method that retrieves derived memories, such as extracted notes or graph nodes, has each of them mapped back to the sources it came from. Duplicates are dropped, keeping the first occurrence, and T_q is the set of the first ten identifiers that remain.

Per-query scores. The two scores used in the paper are

$$r_q^{\text{frac}} = \frac{|G_q \cap T_q|}{|G_q|} \qquad \text{and} \qquad r_q^{\text{all}} = \begin{cases} 1 & \text{if } G_q \subseteq T_q, \\ 0 & \text{otherwise.} \end{cases}$$

The fractional score r_q^{frac} is the share of the gold sources that appear in the top ten. The all-gold score r_q^{all} is one only when all of them do. The two agree whenever $|G_q| = 1$. For a query with three gold sources of which two are retrieved, $r_q^{\text{frac}} = 2/3$ while $r_q^{\text{all}} = 0$.

Averages. With r_q standing for either score, the *query average* is

$$\text{R@10}_{\mathcal{Q}} = \frac{100}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} r_q.$$

On the two persona-centric benchmarks the queries are partitioned by the persona whose history they are asked about. Writing $\mathcal{P}$ for the personas and $\mathcal{Q}_p$ for the queries of persona p, the *persona average* is

$$\text{R@10}_{\mathcal{P}} = \frac{100}{|\mathcal{P}|} \sum_{p \in \mathcal{P}} \frac{1}{|\mathcal{Q}_p|} \sum_{q \in \mathcal{Q}_p} r_q,$$

which gives every persona the same weight regardless of how many queries it has. The two averages coincide when every persona has the same number of queries.

CloneMem: persona average of r^{frac}. Each question carries one or more evidence references, and each reference lists the media items it rests on; G_q is the union of those items. The benchmark reports three variants of Recall@k (Hu et al., 2026, §5.2); the one we use, Recall-Flat, is "the overall proportion of unique ground-truth media IDs retrieved", which is r_q^{frac}. The official scorer computes it per user over the deduplicated top-k list and then averages the ten per-user results with equal weight, which is the persona average. The other two variants, Recall-All-Any and Recall-Any-Any, ask whether at least one item of every evidence reference, or of any reference, is retrieved; neither is used here.

RealMem: persona average of r^{all}. Gold sources are dialogue sessions. The paper reports "Recall@k" without further definition (Bian et al., 2026, §4), and its official scorer computes two quantities, an any-gold score that is one when at least one gold session is retrieved and the all-gold score r_q^{all}. We identify which one the paper's Table 3 reports by rescoring our own runs both ways. For Mem0 the all-gold score is 60.74 and the any-gold score 81.40, against a published 60.55; for A-Mem they are 60.43 and 79.98, against a published 57.60. Only the all-gold score is consistent with the published values, and it is the one we report. The scorer produces one result per persona and does not specify how they are combined; we give the personas equal weight, as CloneMem does. Taking the query average instead changes these two cells by less than 0.2 points (60.57 and 60.42).

NovelQA as packaged by LMEB: query average of r^{frac}. LMEB converts NovelQA into passage retrieval: 1,541 queries over 79,286 passages from 61 books, with 1.63 gold passages per query on average. Each query is ranked only against the passages of its own book, the candidate set LMEB distributes with the data, and we use the same candidate sets. It follows the MTEB evaluation, in which Recall@10 is r_q^{frac} averaged over queries (Zhao et al., 2026). NDCG@10 is LMEB's primary metric; its R@10 values, including the bge-m3 figure in Table 1, are reported in its appendix (Table 18). The corpus is not split into personas, so the only average is over queries.

Why rows are not compared. The three rows differ in the per-query score, in the average, in the kind of source that is retrieved and in the number of candidates each query is ranked against. An all-gold score is never higher than the fractional score on the same ranking: scored fractionally, the RealMem Mem0 cell would rise from 60.74 to 71.00. For these reasons Table 1 is read within rows only.

Cost accounting. Token counts include failed requests and retries and give no discount for cached input. Build and retrieve times exclude process start-up. The comparators and Flat run on a node with 8× NVIDIA B200 GPUs; our memory is built and queried on a single RTX 4090, which also serves the reranker.

E LOCALITY IN PRACTICE

The closure is defined structurally, as three levels around the focal item, and nothing bounds the number of nodes it contains: a parent with many items directly under it yields a large closure. Table 6 therefore measures how much a decision actually reads. We take every snapshot of the 81 trees built for Table 1—1,235 snapshots, taken as each tree grows—and, for up to 200 items per snapshot, count the nodes the two decisions would read around that item: for reorganization, the parent, the grandparent, the item's siblings, its sibling categories and the other children of the grandparent; for attach, the current node and its child categories at every step of the descent from the root. The sample has 157,555 items.

Table 6: Nodes read by one decision, against the number of items N already in the tree. Closure: the window of a reorganization decision; descent: all windows of an attach decision together.

N	Sampled	Closure (median / p90 / max)			Closure/N	Descent (median)
< 100	24,359	33	54	79	51.8%	8
100–300	83,796	45	84	134	28.4%	17
300–1,000	36,400	22	46	243	4.0%	19
1,000–3,000	12,600	25	59	368	2.0%	22
$\geq 3{,}000$	400	30	215	1,484	0.5%	49

As the trees grow by more than an order of magnitude, the median closure stays between 22 and 45 nodes, so its share of the tree falls from about half to 0.5%, and the descent grows only with the depth of the tree. The tail is real: in the largest trees a few closures reach several hundred nodes, and one reaches 1,484. The closure is therefore small in practice, not bounded in principle.

F THE ROUND IN FULL

Algorithm 1 states one round of online construction, as summarized in §3. During labeling both decisions are made by the frozen teacher $\mathcal{O}$ (§3.3); at deployment both are made by the single trained policy of §3.2.

Algorithm 1 Online two-stage tree construction for entity e (rollout)

Require: item stream $\mathcal{F}_e$; policy π_θ serving both decisions (replaced by the teacher $\mathcal{O}$ during labeling, §3.3)
Ensure: tree T

$T \leftarrow \{\text{root}(e)\}$
for $f \in \mathcal{F}_e$ **do** ▷ **Stage 1: attach navigation**
  $v \leftarrow \text{root}(T)$
  repeat
    $a \leftarrow \pi_\theta(\cdot \mid v, f)$ over $\mathcal{N}(v) = \{v\} \cup \{\text{internal children of } v\}$
    if $a = 0$ **then** attach f under v; **break else** $v \leftarrow \mathcal{N}(v)[a]$
  until f is attached
  $c \leftarrow f$
  loop ▷ **Stage 2: local reorganization**
    $C \leftarrow \text{closure}(c)$
    $\{a_i\}_{i \in O},\ \{\tau_i\}_{i:\,\text{DEMOTE}} \leftarrow \pi_\theta(\cdot \mid C)$
    apply the edits (serialized GROUP → DEMOTE → PROMOTE)
    break if T is unchanged, if c has moved, or if a GROUP founded a category
  end loop
end for
return T

G Retrieval Procedure and Tree Factors

This section specifies the retrieval behind the Ours rows of Table 1 and the ablations of Table 2. Every weight is frozen, and the same procedure and weights are used on all three benchmarks.

Candidates. The candidates of a query q are the facts visible to it: on CloneMem and RealMem, the facts of the tree snapshot taken at q's question time; on NovelQA, the facts of q's own book. Write N for their number. A fact that the policy has not filed under any aspect remains a candidate, and every tree factor below assigns it 0. If $N \leq 10$, all candidates are returned.

Floor. Each candidate f receives the score

$$t(f) = \text{mm}\big(\cos(\mathbf{e}_q, \mathbf{e}_f)\big) + 0.20 \cdot \text{mm}\big(\text{BM25}(q, f)\big),$$

where mm is min-max normalization over the N candidates of this query. The embeddings $\mathbf{e}$ are bge-m3 (CLS pooling, ℓ_2-normalized, 2,048-token window, no query instruction). BM25 is computed over the N candidates alone ($k_1 = 1.2$, $b = 0.75$), on lower-cased tokens of at least two word characters with English stop words removed, counting each query term once.

First stage: tree factors. Let P be the 200 candidates with the highest t (all N if fewer), and let the *seeds* $s_1, \ldots, s_5$ be the floor top five. The first stage reorders P by

$$S(f) = t(f) + 2 \cdot \frac{\Delta}{0.05} \cdot \varphi(f), \qquad \varphi(f) = 0.2\, X_1(f) + 0.75\,(0.01 + 0.015\,\bar{d})\, X_2(f) + 0.05\, X_3(f),$$

with ties kept in floor order. Here Δ is the floor score at rank 6 minus that at rank 11, set to 0.05 when it is not positive or when $N \leq 6$; the factor term is thus measured in units of the query's own floor spread just below the top five. X_k is the raw score of factor Fk min-max-normalized over P, and $\bar{d}$ is the mean number of aspect ancestors of the N candidates (unfiled facts count 0), so that F2 weighs more in deeper trees. These values are frozen and the same for every benchmark.

Reranker. The first-stage top 40 are scored one at a time by Qwen3-Reranker-4B, which sees only the query and the text of the candidate and no tree information. The prompt asks whether the record is relevant to the question, and the score is $\log p(\texttt{yes}) - \log p(\texttt{no})$ for the first generated token under greedy decoding. Candidates are truncated to 4,096 tokens and queries to 1,024. The final top 10 is ordered by this score alone, ties going to the better first-stage rank.

The tree factors. The factors read the tree snapshot of the query: the parent–child structure, each fact's aspect ancestors, the arrival order of the facts and, for F2, the description of every aspect node through its bge-m3 embedding $\mathbf{n}_v$ (2,048-token window). All three also read the floor ranking: F1 and F2 the five seeds, F3 the floor top 10. None reads the dataset identity, the question type or the gold evidence, and the query enters only through $\mathbf{e}_q$ and through the floor. Write $\mathrm{pos}(f)$ for the arrival rank of f, $d_r(f) = |\mathrm{pos}(f) - \mathrm{pos}(s_r)|$ for the seed s_r of floor rank r, $\mathrm{leaf}(f)$ for the aspect f is filed under, $\mathrm{top}(f)$ for its top-level aspect and $\mathrm{depth}(f)$ for its number of aspect ancestors; unfiled facts score 0 in every factor. F2 scores an aspect node v by how much better its name matches the query than it matches the scope on average,

$$g(v) = \cos(\mathbf{n}_v, \mathbf{e}_q) - \frac{1}{N}\sum_f \cos(\mathbf{n}_v, \mathbf{e}_f).$$

- **F1 aspect continuation** (`aspect_continuation`, targeting C1). The item filed right next to a top-ranked item, in the same leaf aspect at least two levels deep:

$$X_1(f) = \max_r \; r^{-1/2}\, \frac{\mathbf{1}[\mathrm{leaf}(f) = \mathrm{leaf}(s_r),\; 1 \le d_r(f) \le 3,\; \mathrm{depth}(s_r) \ge 2]}{1 + d_r(f)},$$

  and 0 if no seed qualifies.
- **F2 event context** (`F2_K2`, targeting C2). $X_2(f) = M(f) + N(f) + 0.25\,L(f)$. The moment term $M(f) = \max_r \mathbf{1}[d_r(f) \le 20,\; \mathrm{top}(f) = \mathrm{top}(s_r)]/(1 + d_r(f))$ credits items that arrived near a seed under the same top-level aspect (a filed seed scores 1). The named term $N(f) = 1$ if $\mathrm{leaf}(f)$ is one of the two aspect nodes of the snapshot with the highest $g(v)$ and lies on the ancestor chain of a seed. The leaf term $L(f) = 1$ if $\mathrm{leaf}(f) = \mathrm{leaf}(s_r)$ for some filed seed, at any distance.
- **F3 thread tip** (`F3_thread_tip`, targeting C3). At each of two levels, the top-level aspect and the parent of the leaf (the top-level aspect when the depth is 1), take the floor top 10 of P. For each aspect v at that level that holds at least one of them, let c_v be how many of the 10 it holds and first_v the earliest arrival among them. If v holds one of the 30 newest facts of the snapshot, the 2 newest candidates in P filed under v that arrived after first_v score $c_v/10$ (maximum over v). X_3 is the maximum over the two levels of these values, each min-max-normalized over P.

Table 7: Question classes, by where the gold evidence lies, and the tree factor that targets each.

Class (gold evidence)	Factor	What the factor promotes
C1 lookup (one or few records)	F1 aspect continuation	items filed next to a top-ranked item in its own deep category
C2 multi-evidence (spread records)	F2 event context	items near a top-ranked item in arrival and branch, in its category, or under the best-named category holding it
C3 latest state (newest records)	F3 thread tip	the newest items of the categories that hold the top-ranked items

How the factors were designed and set. The factors were designed together with an LLM coding agent on the questions and trees of LoCoMo, our development set; the three benchmarks of Table 1 were not used to design them. Because each factor is meant to serve one class of question rather than retrieval in general, the agent was told in advance which question classes the benchmarks contain, with one example question of each; without knowing the classes it could not have designed a factor for them. The frozen values above were then set by a separate agent in a single run and were not revised afterwards, so neither the factors nor their weights were tuned repeatedly against the test questions.

Table 8 runs the frozen pipeline on LoCoMo and on LongMemEval, whose trees we built the same way, and reports what the tree factors add on top of cosine similarity, BM25 and the reranker. On LoCoMo they add $+0.93$, most of it from F1 and F2; carried unchanged to the three test benchmarks, they add $+0.92$, $+0.25$ and $+0.18$. On LongMemEval they bring almost nothing, which we discuss in §4.4.

Table 8: The frozen pipeline on LoCoMo, where the factors were designed, and on LongMemEval (query-averaged fractional R@10). Depth: mean number of aspect ancestors of a fact in our trees. Δ: what the tree factors add on top of cosine + BM25 + reranker, with its 95% paired-bootstrap interval.

Dataset	Qs	Depth	Cos	+BM25+RR	+factors	Δ	95% CI
LoCoMo (development)	1,982	2.14	64.46	79.73	80.67	$+\mathbf{0.93}$	$[+0.43, +1.47]$
LongMemEval, turns	470	1.05	89.74	95.54	95.54	$+0.00$	$[+0.00, +0.00]$
LongMemEval, sessions	470	0.54	96.27	98.94	99.01	$+0.07$	$[-0.07, +0.32]$

Question classes. The per-class columns of Table 2 group each benchmark's native question types by where the gold evidence lies in the memory, not by the type's name. **C1 lookup** (abstract-lookup): the answer rests on a single record or on very few. **C2 multi-evidence** (aggregation): several gold records spread across the arrival stream have to be combined. **C3 latest state** (recency): the gold is the newest part of the memory, with at least 30% of a question's gold records in the last 10% of the arrival stream and their mean position at 0.65 of the stream or later. Table 9 gives the assignment; the factors never read it.

Table 9: Assignment of native question types to classes, with the number of questions.

Class	CloneMem	RealMem	NovelQA
C1 lookup	unanswerable (109)	Temporal-reasoning (34)	Character (271), Meaning (265), Plot (333), Relation (125), Setting (154), Span (25)
C2 multi-evidence	comparison (179), counterfactual (188), inference (125), pattern (194), trajectory (190)	NA	Times (368)
C3 latest state	causal (202)	Dynamic Incremental (1,002), Dynamic Updating (195), Implicit Preference (184)	NA

H PROMPTS

This section lists every prompt an LLM receives in our pipeline: the teacher that labels the training data (§3.3), the input and output format of the 0.5B policy, the model that names new categories, the reranker used at retrieval, and the judge used to evaluate the teacher and the policy. Each prompt is given verbatim; text in angle brackets, such as `<NEW FACT TEXT>`, marks where content is inserted. Typographic characters (dashes and the box-drawing characters of one diagram) are rendered in ASCII. The prompts were fixed for the whole of each run.

H.1 TEACHER: ATTACH DECISION

The teacher (Qwen3.8-27B) receives a system and a user message and answers without thinking mode.

Teacher, attach: system message

```
You are a knowledge tree builder. Given a new fact and candidate parents, choose the best
    parent index.
Choose the MOST SPECIFIC (deepest) appropriate parent. Prefer semantic relevance over breadth
    .
Index 0 = the CURRENT node (choosing it means: attach at THIS level, do not descend into a
    child) -- only choose it when no listed child aspect is a better match. NOTE: index 0 is
     the entity root only at the top level; once you have descended, index 0 is the current
    aspect, not the root.

Before answering, check in this order:
```

```
1. Is the fact a GENERAL description of the current node itself? -> index 0
2. Is the fact incidental trivia (anecdote, ambience, aside) whose TOPIC matches no listed
   aspect? -> index 0. Physical containment (it happens there) is NOT topical relevance.
3. Otherwise pick the MOST SPECIFIC listed aspect whose TOPIC covers the fact.
Most facts fall in 1 or 2 - do not force a fact into a loosely-related aspect.

Answer in EXACTLY two short lines, nothing else:
Reason: <one short sentence naming the candidate you chose and why>
Final Answer: <index>
```

Teacher, attach: user message

```
New fact (id=<FACT_ID>): <NEW FACT TEXT>

Candidates:
[0] <CURRENT NODE NAME: entity root or current aspect> (<root|aspect>)
[1] <CHILD ASPECT 1 NAME> (aspect)
[2] <CHILD ASPECT 2 NAME> (aspect)
[3] <... one line per child aspect ...> (aspect)

Decision rules (MANDATORY -- you must follow these exactly):
- HARD RULE: If ANY aspect in the candidate list is topically related to the new fact (even
     partially), you MUST choose that aspect index. You are NOT allowed to choose index 0 (
     the current node) when a topically relevant child aspect exists. This is not a guideline
      -- it is a rule.
- When multiple aspects are topically relevant, choose the MOST SPECIFIC (deepest) one that
     covers the new fact's topic.
- Only choose index 0 (the current node) if BOTH conditions hold: (a) the fact belongs at
     THIS level (no listed child aspect is a deeper, better fit), AND (b) NO aspect in the
     candidate list is topically relevant.
- Incidental trivia (pets, decor, small anecdotes, ambience) whose TOPIC matches no listed
     aspect MUST stay at index 0 -- physical containment (it happens at this place) is NOT
     topical relevance. Do not force such facts into a loosely-related aspect.

End your response with a single line: Final Answer: <index>
```

H.2 TEACHER: REORGANIZATION DECISION

The reorganization prompt is answered in thinking mode; only the final answer is parsed. The user message is the closure serialized as JSON. When an answer fails validation, the same request is sent again with the validator's error appended (Appendix B); this happens on fewer than 0.1% of calls.

Teacher, reorganization: system message

```
=== 1. WHAT YOU ARE WORKING ON ===

There is a MEMORY TREE that organises a large number of facts about one entity into a
hierarchy. The tree has only two kinds of node:

- **fact**: one indivisible piece of source text, e.g. The company was founded in
  Beijing in 2012. A fact is always a LEAF - nothing ever hangs under it.
- **aspect**: a category node that has ONLY A NAME AND NO CONTENT of its own, e.g.
  Founding and Funding. Its job is to give a group of related facts a topic name.
  **An aspect can contain other aspects**, and that is what gives the tree its depth.

A good tree puts every fact under the MOST SPECIFIC aspect that genuinely covers it,
and creates an aspect only when several facts really do share one theme.

You never see the whole tree. Each time you are given one small slice cut out of it,
called a **closure** - a three-layer local view. You decide only about this slice;
the rest of the tree is not your concern.

=== 2. WHAT THIS CLOSURE LOOKS LIKE ===

```
root --------------------------- the top of THIS window. It is NOT necessarily the
| root of the whole tree - very likely it is itself
| an aspect buried deep inside a much larger tree.
|
+-- parent * THIS IS WHAT YOU ARE REORGANISING *
| |
| +-- fact (1)   +
| +-- fact (2)   +- together these are `operable_facts`
| +-- fact (3) | **you may act ONLY on these, and on all of them**
| +-- fact (4)   +
| |
```
```

```
|   +-- aspect sub-category A   +
|   +-- aspect sub-category B   +- parent's existing sub-aspects = `valid_down_targets`
|                                    **DOWN can only move a fact into one of these**
|
+-- aspect uncle category X     +
+-- aspect uncle category Y     +- parent's SIBLING aspects
|                                    = `context_only.aspects_elsewhere_NOT_down_targets`
|                                    (!) outside your scope, **NOT legal DOWN targets**
|
+-- fact (5)                      +
+-- fact (6)                      +- facts hanging directly under root
                                     = `context_only.facts_one_level_up`
                                     (!) you cannot act on them, but **UP is exactly
                                     the move that sends a fact to this level**
```

A concrete example:

```
root      Fruit
|
+-- parent Citrus                 <- reorganising here
|   (1) Chinese oranges are mainly grown in southern Jiangxi
|   (2) Oranges are rich in vitamin C
|   (3) The sourness of lemons comes from citric acid
|   (4) Fruit keeps longer under refrigeration    <- this is about ALL fruit, too general
|   +-- aspect Oranges                          <- valid_down_targets
|   +-- aspect Lemons
|
+-- aspect Berries                <- uncle aspect, NOT a legal DOWN target
+-- (5) Fruit is a major source of vitamins       <- facts_one_level_up
```

A sensible answer here: (1) and (2) are both about oranges -> DOWN into Oranges; (3) is about lemons -> DOWN into Lemons; (4) is about fruit in general, one level above Citrus -> UP.

=== 3. HOW THIS WINDOW FITS INTO THE BIGGER PROCESS ===

The tree grows ONE FACT AT A TIME, and this window is just one step of that process. Two things follow from that:

- **After this round the tree is updated, the window is rebuilt, and you will be asked again.** So do not worry about not finishing in one shot - whatever you leave will come back to you next round.
- `rules_for_this_window` is COMPUTED FOR THIS WINDOW: `up_is_legal`, `down_is_legal`, `must_output_exactly_n_actions`. **Read it; do not derive it yourself.** (For instance, when `parent` happens to BE `root`, there is no level above, so `up_is_legal` is false.)

=== 4. WHAT YOU MUST DO ===

Give exactly one action to EVERY fact in `operable_facts`:

- **IDLE**: it is already in the right place, leave it alone.
- **DOWN**: the fact is too specific for `parent`; move it into an EXISTING sub-aspect. `target_aspect` MUST be copied VERBATIM from `valid_down_targets` - no rewording, no abbreviation, no case change. **DOWN can NEVER create an aspect.** If the home it needs does not exist yet, your choice is GROUP or IDLE - never DOWN with an invented name.
- **UP**:   the fact is too general for `parent`; it describes the level above. Move it up one level, alongside `facts_one_level_up`. Legal only when `rules_for_this_window.up_is_legal` is true.
- **GROUP**: at least 3 facts share a clear, specific theme; create a new sub-aspect for them. A group with fewer than 3 members will be rejected.

>>> You may create AT MOST ONE group per response. <<<

This is a hard requirement, not a preference:
- Never output two groups at once (e.g. grouping 1,2,3 AND also grouping 4,5,6). If you see several groupable themes, output only the STRONGEST one. The remaining facts do not join a group, but they MAY still take another action.
- Facts outside the group are free to be DOWN / UP / IDLE. They must not overlap with the group: a fact is either a group member or it takes one of the other actions - **never both**.

When to create a group depends on the parent's current shape:

- The parent has several child facts that are semantically very close to each other -> group them, leave the rest IDLE
- The parent has too many flat facts -> this layer is overloaded: create one group for

```
  the tightest theme, and in the same round move the obvious DOWN candidates into the
  existing sub-aspects
- The parent has few facts -> do not create a group; use IDLE / DOWN only

**Grouping is the EXCEPTION, not the default.** Most rounds create no group at all,
and your one group must stay well under half the layer - packing most of the layer
into one group merely renames it, and will be rejected.

A clear, specific theme means the same entity, the same event, or the same concrete
concept. Merely sharing a generic word (e.g. AI, company) is NOT enough.
When torn between a weak GROUP and IDLE, choose IDLE.

=== 5. OUTPUT FORMAT ===

One JSON object per fact, one per line, nothing else.
**No prose before or after, no ``` fence.**

{operation: IDLE,  fact_id: <id>}
{operation: DOWN,  fact_id: <id>, target_aspect: <verbatim copy>}
{operation: UP,    fact_id: <id>}
{operation: GROUP, member_ids: [<id>, <id>, <id>], new_aspect: <new aspect name>}

=== 6. LEGALITY CHECKLIST ===

If any line breaks one of these, the WHOLE response is thrown away and you are asked
again. Everything you need is printed in the payload - never use an id or a name that
is not there.

1. **fact_id**: the only legal ids are the ones listed in `operable_facts`. Everything
   under `context_only` is read-only; never act on it.
2. **Coverage**: every operable fact is covered exactly once. A GROUP line already
   covers all of its `member_ids`; every remaining operable fact still needs its own
   line. Never give the same fact two lines. The total must account for exactly
   `rules_for_this_window.must_output_exactly_n_actions` facts.
3. **DOWN target**: copy one string from `valid_down_targets` character-for-character.
   That list is the complete set of sub-aspects that exist right now.
4. **UP**: legal only when `rules_for_this_window.up_is_legal` is true.
5. **GROUP**: at least 3 members, and it must not cover most of the layer.
6. **new_aspect**: must be a new, more specific name. It may not repeat `parent` or
   `root`.

=== 7. BEING CAUTIOUS IS NOT SAFER ===

Answering all-IDLE for a layer that clearly needs work is just as wrong as answering
illegally - it silently stalls the tree, and nothing tells you it went wrong. The
checklist above tells you which line is LEGAL; it says nothing about how conservative
to be.

When a fact clearly belongs under one of `valid_down_targets`, say DOWN. When it
clearly describes the level above `parent`, say UP. Keep IDLE for the facts that
genuinely belong exactly where they already are.
```

Teacher, reorganization: user message

```
The tree window is:
{
  entity_name: <name of the true entity root of the whole tree>,
  depth_from_entity_root: <int: hops from window root to entity root; -1 = unknown>,
  root: <window root description>,
  parent: <parent aspect description (the node being reorganised)>,
  parent_is_root: <true|false>,
  rules_for_this_window: {
    up_is_legal: <true iff parent is not root>,
    down_is_legal: <true iff valid_down_targets is non-empty>,
    must_output_exactly_n_actions: <K = number of operable_facts>
  },
  operable_facts: [
    {
      id: <fact id>,
      text: <FACT TEXT>
    },
    ... one object per operable fact (center fact + facts directly under parent; no center
     marker) ...
  ],
  valid_down_targets: [
    <existing sub-aspect name of parent>,
    ...
  ],
  context_only: {
    facts_one_level_up: [
```

```
      {
        id: <fact id>,
        text: <FACT TEXT directly under root, not operable>
      },
      ...
    ],
    aspects_elsewhere_NOT_down_targets: [
      <sibling aspect of parent (uncle), not a legal DOWN target>,
      ...
    ]
  }
}
```

H.3 POLICY INPUT AND OUTPUT

The 0.5B policy reads the same states in a compact format and writes its answer as JSON; the grammar of the answer is the one described in §3.2.

Policy, attach: system message

```
You are a tree-placement policy. Select exactly one legal candidate parent for the new fact.
    Candidate IDs are local and carry no meaning. Return only the required JSON object.
```

Policy, attach: user message

```
{candidates:[{id:C0,legal:true,role:<current|child>,text:<CANDIDATE TEXT>},{id:C1,legal:true,
    role:<current|child>,text:<CANDIDATE TEXT>},...],new_fact:<NEW FACT TEXT>,
    output_contract:{selected_parent:one legal candidate id},schema_version:policy-sft-v1,
    task:stage1_parent_selection}
```

Policy, attach: answer

```
{selected_parent:<one legal candidate id, e.g. C2>}
```

Policy, reorganization: system message

```
You are a local tree-reorganization policy. Give exactly one action for every operable fact.
    Fact and target IDs are local and carry no meaning. Return only the required JSON object
    .
```

Policy, reorganization: user message

```
{down_targets:[{id:A0,text:<EXISTING SUB-ASPECT NAME>},...],operable_facts:[{id:F0,text:<FACT
     TEXT>},{id:F1,text:<FACT TEXT>},...],output_contract:{actions:exactly one keyed action
    for every operable fact,down_target:one down_targets id when op is DOWN,group:all GROUP
    facts use G0,operations:[IDLE,UP,DOWN,GROUP]},parent:<PARENT ASPECT NAME>,root:<WINDOW
    ROOT NAME>,schema_version:policy-sft-v1,task:stage2_local_reorganization}
```

Policy, reorganization: answer

```
{actions:{F0:{op:IDLE},F1:{op:DOWN,target:A0},F2:{group:G0,op:GROUP},...}}
```

H.4 CATEGORY NAMING

When a GROUP founds a new category during tree building, gpt-4o-mini names it from the grouped items.

Category naming: system message

```
You are a tree organization expert. Given a list of facts, generate a concise category name
    for them. Output ONLY the category name (1-5 words), no quotes, no explanation.
```

Category naming: user message

```
Given the following facts that will be grouped together:
- <MEMBER FACT 1 TEXT (full, untruncated)>
- <MEMBER FACT 2 TEXT>
- <... one line per GROUP member ...>
```

```
The group will become a NEW sub-category under the parent category <PARENT CATEGORY NAME>.
Existing category names (parent and its current sub-categories):
- <PARENT CATEGORY NAME>
- <EXISTING SUB-CATEGORY 1>
- <EXISTING SUB-CATEGORY 2>

The new name must be DIFFERENT from all of these -- not equal and not a trivial paraphrase of
      any of them.

Generate a short, concise description (1-5 words) for this group. Output ONLY the description
      text, no quotes or explanation.
```

H.5 RERANKER

Qwen3-Reranker-4B scores each query–candidate pair from a single completion prompt; the score is $\log p(\texttt{yes}) - \log p(\texttt{no})$ of the first generated token (Appendix G).

Reranker prompt

```
<|im_start|>system
Judge whether the Document meets the requirements based on the Query and the Instruct
     provided. Note that the answer can only be yes or no.<|im_end|>
<|im_start|>user
<Instruct>: Given a question and one candidate record from a long-term memory store, judge
     whether the record is relevant to the question.
<Query>: <QUERY>
<Document>: <DOCUMENT><|im_end|>
<|im_start|>assistant
<think>

</think>
```

H.6 JUDGE

The accepted-answer evaluation of §3.3 and §4.1 gives gpt-6-sol one prompt per decision type; gpt-6-luna answers the same prompts. The reasonableness check of Appendix B uses a separate rubric.

Judge, attach: system message

```
You are placing a new fact into a memory tree.

The fact is being added under one node. You are shown that node (the one it sits at now)
and the sub-aspects that already exist under it. Choose exactly one home for the fact:

- one of the sub-aspects, if the fact clearly belongs to that specific theme;
- the node it sits at now, if no existing sub-aspect covers it, or if the fact is
  general enough to belong at that level.

Candidate ids (C0, C1, ...) are local labels and carry no meaning; judge only by the text.

First give your answer. Then -- separately and honestly -- list every OTHER candidate that
a careful expert could also defend for this fact. Most placements have exactly one good
home; say NONE when that is the case. Do not pad the list, and do not hide a genuine tie.

Answer in exactly three lines:
ANSWER: C<k>
ALSO_ACCEPTABLE: C<i>, C<j>   (or NONE)
WHY: <=20 words
```

Judge, attach: user message

```
NEW FACT:
  <NEW FACT TEXT, cut to 700 chars +  ...[truncated]>

WHERE IT COULD GO:
  THE NODE IT SITS AT NOW:
    C<k>: <CURRENT NODE TEXT, cut to 700 chars>
  SUB-ASPECTS THAT ALREADY EXIST UNDER THAT NODE:
    C<i>: <CHILD ASPECT TEXT>
    ... (one line per child aspect)
```

Judge, reorganization: system message

```
You are deciding what should happen to ONE fact in a memory tree.

The system keeps a tree of facts under aspect nodes.

- IDLE  : the fact already sits in the right place; leave it.
- DOWN  : the fact is too specific for the PARENT; move it into one of the EXISTING
          sub-aspects listed in down_targets. DOWN can never invent a new aspect.
          If down_targets is empty, DOWN is illegal.
- UP    : the fact is too general for the PARENT; it describes the level above.
- GROUP : at least 3 facts under the PARENT share one clear, specific theme; open a NEW
          sub-aspect for them. A GROUP must not cover most of the level.

House rule: when torn between a marginal GROUP and IDLE, choose IDLE.

You are given the state of one PARENT node: the facts sitting directly under it and the
sub-aspects that already exist under it. Decide the correct operation for the ONE fact
named below.

First give your answer. Then -- separately and honestly -- list every OTHER operation that
a careful expert could also defend for this fact. Most cases have exactly one right answer;
say NONE when that is the case. Do not pad the list, and do not hide a genuine tie.

Answer in exactly three lines:
ANSWER: IDLE | UP | DOWN:<the exact down_target id> | GROUP
ALSO_ACCEPTABLE: <other operations, same format>   (or NONE)
WHY: <=20 words
```

Judge, reorganization: user message

```
ROOT: <ROOT TEXT>
PARENT: <PARENT TEXT>

FACTS DIRECTLY UNDER THE PARENT:
  F0: <FACT TEXT, cut to 700 chars +  ...[truncated]>
  F<j>  <-- THE FACT TO DECIDE: <FACT TEXT>
  ... (one line per operable fact)

EXISTING SUB-ASPECTS (the only legal DOWN targets):
  A0: <SUB-ASPECT TEXT>
  ...            (or the single line   (none))

Decide the operation for fact F<j>.
```

Reasonableness check: system message

```
You are auditing ONE decision made by an automatic memory-organising policy.

The policy maintains a tree of facts under aspect nodes. It looks at one aspect (the PARENT),
the facts sitting directly under it, and the aspects that already exist under it, and it
     chooses
one operation for each fact:

- IDLE  : the fact already sits in the right place; leave it.
- DOWN  : the fact is too specific for the PARENT; move it into one of the EXISTING sub-
     aspects
          listed in `valid_down_targets`. DOWN may never invent a new aspect.
- UP    : the fact is too general for the PARENT; it describes the level above.
          Legal only when `up_is_legal` is true.
- GROUP : at least 3 facts under the PARENT share one clear, specific theme; create a NEW
          sub-aspect for them.

House rules the policy was given:
- When torn between a marginal GROUP and IDLE, prefer IDLE.
- A GROUP must not cover most of the level.
- Answering IDLE for a fact that clearly belongs in an existing sub-aspect is a real error,
  not a safe default.

Rate the ONE decision you are shown on this three-point scale:

- CLEARLY_RIGHT : this is what the rules call for; a careful expert would do the same.
- DEFENSIBLE    : not the only good answer, but a reasonable reading of this state. An expert
                  might choose otherwise yet could not call this one wrong.
- WRONG         : it breaks a rule, or it misreads what the fact is about.

Be strict. Mark WRONG whenever any of these holds:
- DOWN into a sub-aspect whose topic does not cover this fact;
- DOWN when `valid_down_targets` is empty, or into something not in that list;
```

```
- UP when `up_is_legal` is false;
- GROUP with fewer than 3 members, or whose members do not share one clear specific theme;
- IDLE when the fact plainly belongs in one of the listed `valid_down_targets`.

Do NOT rate a decision DEFENSIBLE merely because some argument for it can be imagined.
DEFENSIBLE is for genuine ties between readings of the same state.

Answer in exactly three lines:
RULE: <the rule that decided it, <=12 words>
VERDICT: CLEARLY_RIGHT | DEFENSIBLE | WRONG
WHY: <=25 words
```

Reasonableness check: user message

```
STATE:
<STATE JSON = json.dumps(slim_state(...), ensure_ascii=False, indent=1); keys: root, parent,
    up_is_legal,
 depth_from_entity_root, facts_under_parent(operable) [id,text cut to 900 chars],
    valid_down_targets,
 sibling_aspects_of_parent, other_facts_in_window [<=12, text cut to 400 chars]>

The fact under audit is the one with id <FACT_ID>:
  <FACT TEXT, first 1200 chars>

DECISION: <DECISION SENTENCE, one of:
  Leave this fact where it is (IDLE).
  Move this fact UP to the level above the parent (UP).
  Move this fact DOWN into the existing sub-aspect: '<NAME>' (DOWN).
  Bundle this fact together with facts [id <a>, id <b>, ...] into a NEW sub-aspect (GROUP).>
```