UNIVERSITÉ DU QUÉBEC EN OUTAOUAIS

DÉPARTEMENT DES SCIENCES ADMINISTRATIVES

WORKING PAPER NO. 30

Beyond Text: Cache Compression and Knowledge Transfer in Multi-Agent Systems

A Reproducible Study of Fidelity, Communication Cost, and Repeated Handoffs

Simon-Pierre Boucher

Département des sciences administratives
Université du Québec en Outaouais

simon-pierre.boucher@uqo.ca

Gatineau – Pavillon Alexandre-Taché
283, boulevard Alexandre-Taché
Gatineau, Québec, Canada J9A 1L8

October 2026

Version 1.0

Abstract

Abstract to be written after the results.

Keywords: KV cache compression; cache reuse; multi-agent systems; knowledge transfer; prompt compression; structured evidence; reproducibility; Apple Silicon; MLX

JEL Classification: C63, C88, D83, L86, O33

Contents

1 Introduction **5**

2 Related work **6**

3 Problem formulation and taxonomy **7**

3.1 Protocol notation 7

3.2 Accounting 8

3.3 Taxonomy 9

3.4 Research questions and registered hypotheses 9

4 Methods **11**

4.1 Track A: text, structured messages and evidence 11

4.2 Track B: within-model KV compression and reuse 12

4.3 Track C: cross-checkpoint transfer 13

4.4 Track D: hosted receivers 13

5 Experimental protocol **14**

5.1 Models, hardware and runtimes 14

5.2 Task families and splits 14

5.3 Cache correctness before performance 15

5.4 Pilot, freeze and campaign 16

5.5 Statistics 16

6 Results **16**

7 Adaptive **16**

8 Limitations **16**

9 Reproducibility **17**

10 Conclusion **17**

Appendices **20**

A Registered hypotheses **20**

B Source audit and unexecuted methods **21**

C Correctness tests **21**

D Prompts **22**

E Pseudocode **23**

F Per-attempt record schema **23**

G Additional tables **24**

1. Introduction

Agents built on large language models rarely work alone. A reader agent summarizes a filing for an analyst agent; a crawler hands extracted facts to an aggregator; a long document is processed once and then queried many times by different workers. Every such handoff chooses a *representation* for the information that crosses the boundary between two model executions: the raw text, a shortened text, a prose summary, a structured record with provenance, a pointer to shared memory, or—when the two executions share a checkpoint—the attention state (the key–value, or KV, cache) that the first model built while reading. A growing literature proposes to compress these states (Liu et al., 2024; Li et al., 2024; Cai et al., 2024; Zhang et al., 2023; Xiao et al., 2024; Zweiger et al., 2026), to reuse them across requests, and, most recently, to translate them between different checkpoints or model families so that a receiver can "inherit" a sender's reading without re-reading the source (Fu et al., 2026; Zou et al., 2026; Alabdulkareem et al., 2026).

The engineering appeal is clear: a receiver that loads a cache skips the prefill of the document, and a compressed cache shrinks memory. The scientific question is less clear. Compression methods are evaluated with different models, different information-access assumptions (does the compressor see the question?), different budgets (tokens, positions, bits, bytes), and different measurement boundaries (kernel time versus end-to-end time, resident tensors versus serialized payloads). Text baselines are often weak or absent. Cache sizes are usually quoted in positions or bits per value rather than in the bytes that actually cross a network, and the receiver side of a handoff—deserialization, loading, fresh-query processing—is seldom counted. Nor is the behaviour of these representations studied over *repeated* handoffs, where small distortions compound.

This paper asks a single question under a common measurement harness: *under which task, model, cache, network and reuse conditions does each communication representation preserve useful information at the lowest end-to-end resource cost?* We answer it empirically, with experiments executed from a single research operator (Claude Code, an agentic coding assistant) on commodity Apple-Silicon hardware, using the MLX runtime, whose KV caches are ordinary tensors that can be inspected, sliced, quantized, serialized and loaded back. All quality numbers come from real model generations; all byte counts are measured on serialized payloads; all latencies are wall-clock spans with device synchronization; and the complete per-attempt record of every trial is released.

Contributions.

1. An **experimentally grounded taxonomy** of communication representations (Table 2) that separates text compression, KV compression, cache reuse, cache transfer and parametric transfer, and records for each method its question-access regime, its compatibility requirements and whether it composes across sources.

2. A **common harness** (`wp30`) with a validated cache adapter layer (identity round trip, prefix equivalence, position preservation, packed-byte accounting, manifest-checked loading), a per-attempt record schema, deterministic scorers, evaluator-isolated answer keys, and document-level cluster bootstrap statistics.
3. **Quality/resource trade-off maps** for twenty-two conditions on two 3B-parameter model families and three task families (synthetic private-fact ledgers, public long-document QA, executable tasks), under question-blind and question-aware regimes, with measured bytes, latency decompositions and peak memory.
4. A study of **repeated handoffs** (1 to 8 transfers) with an evaluator-only fact ledger that measures retention, unsupported additions and the cost of metered repair, and a **fan-in** study with conflicting evidence.
5. **Genuine two-node cache transfer** measurements on a 1 GbE LAN against remote text prefill and local recompute, with a measured break-even model.
6. A **same-family learned KV mapper** (Qwen2.5-1.5B $\rightarrow$ Qwen2.5-3B) trained on train-split caches only, evaluated against text handoffs and an identity-transfer negative control; and a **development-fitted adaptive router** evaluated on held-out data against the best development-selected static method.

Several results are negative or conditional, and we report them as such. Cross-family cache transfer was *not* executed: the only published cross-family method with code (Alabdulkareem et al., 2026) has no released checkpoint and no licence, and our budget did not permit training a mapper of its size; the study is therefore a *partial* transfer study in that respect (Section 8). We also do not claim that any receiver *learned* anything: every effect measured here is contextual, i.e., it vanishes when the message or cache is removed.

2. Related work

Text and prompt compression. Shortening the text a receiver reads is the oldest form of inter-agent compression. Beyond truncation and extractive selection, learned compressors drop tokens with a classifier trained on distilled data (Pan et al., 2024); LLMLingua-2 is task-agnostic by construction, which places it in our *question-blind* regime. Abstractive summaries and structured extraction are the dominant representations in deployed agent frameworks; we treat them as first-class baselines rather than as foils, tuning their prompts on development data only.

KV-cache quantization, eviction and compaction. KIVI (Liu et al., 2024) quantizes keys per channel and values per token to 2 bits with a full-precision residual window, motivated by outlier channels in keys. Eviction methods select positions to keep: H_2O (Zhang et al., 2023) accumulates

attention mass during prefill; SnapKV (Li et al., 2024) scores prefix positions by the attention of an observation window at the end of the prompt, pooled and selected per head; PyramidKV (Cai et al., 2024) allocates decreasing budgets to higher layers; StreamingLLM (Xiao et al., 2024) keeps attention sinks and a recent window and *re-assigns* positions within the cache. Attention Matching (Zweiger et al., 2026) compacts a cache into fewer positions by solving for keys and values that reproduce attention outputs on reference queries. Our audit (Appendix B) records for each method whether its selection observes the question: SnapKV- and PyramidKV-style selection are question-aware by construction when the prompt ends with the question, which matters for the comparison with reusable, question-blind messages.

Cache reuse and transfer. Prefix caching reuses an identical token prefix within one checkpoint; it is exposed by local runtimes (Apple ML Explore contributors, 2026; Hugging Face, 2026) and, as a billing abstraction, by hosted APIs (Anthropic, 2026). Transfer across checkpoints requires a mapping: Cache-to-Cache (Fu et al., 2026) trains a fuser that injects a sharer's cache into a receiver that still prefills the source; LatentMAS (Zou et al., 2026) exchanges full caches and latent thoughts among agents that share the same architecture; SwaT (Alabdulkareem et al., 2026) trains a cross-family translator and combines it with compaction, but releases no checkpoint. Concurrent work on cross-model cache bridging (see the audited list in `literature/sources.csv`) is active; we make no novelty claim about shared latent spaces and run only a same-family linear mapper as a bounded, fully disclosed experiment.

Evaluation. LongBench (Bai et al., 2024) supplies public long-document QA with F1 scoring; RULER (Hsieh et al., 2024) generates controlled retrieval tasks. We combine both ideas: public subsets (QASPER, 2WikiMultihopQA) for external validity, and a new procedural generator of private facts (identifiers, amounts, dates, negations, timestamped revisions, unanswerable questions) so that communication dependence can be tested without pre-training contamination.

3. Problem formulation and taxonomy

3.1. Protocol notation

Let d be a source document, q a downstream query, $y^\star$ its reference answer, S the sender model and R the receiver model. A communication method c under a resource setting b and a question-access policy a produces a message

$$m = E_{c,b}(d; S, a), \qquad \hat{y} = G_R\big(q, D_c(m), \mathscr{A}\big), \tag{1}$$

Table 1. Terminology used in this paper.

Term	Operational meaning	Error avoided
Text compression	Shorter text consumed by a receiver	Calling it KV-cache compression
KV compression	Quantization/selection/transformation of attention tensors	Measuring only a shorter prompt
Cache reuse	Reuse of a compatible previously computed prefix state	Assuming one family ⇒ compatible
Cache transfer	Moving/mapping cache state for a receiver	Sending a summary and calling it latent transfer
Knowledge transfer	Better access to task information via a message	Claiming a frozen receiver learned a skill
Latent communication	An internal representation crosses executions	Calling retrieval embeddings a KV state
Shared memory	Addressable external records under a protocol	Treating an identifier as free content access
Prefill-free handoff	Receiver does not prefill the transferred document	Claiming the fresh query needs no processing
Evidence traceability	Output linked to a resolvable source span	Treating self-reported confidence as proof

where D_c denotes decoding, loading or mapping at the receiver and $\mathscr{A}$ is the explicitly allowed artifact or retrieval interface (empty for all methods except A6, the reference-plus-retrieval method, and the repairable chains of Section **??**). Equation (1) is notation for a protocol, not a claim that methods share an implementation: for text methods m is a string the receiver prefills; for cache methods m is a set of tensors installed as the receiver's attention state, and only the fresh query tokens are prefilled.

Question access. Under the *question-blind* regime the sender observes d and a generic instruction but not q; the message is reusable across the questions of a document. Under the *question-aware* regime both sides may use q. The two regimes are reported separately throughout; a selection method that observes the question (SnapKV-style) is never compared as if it were blind.

Terminology. We follow a fixed vocabulary (Table 1). In particular, *text compression* is a shorter text consumed by a receiver; *KV compression* is quantization, selection or transformation of actual attention tensors; *cache reuse* reuses a compatible, previously computed prefix state; *cache transfer* moves or maps a state for a receiver; and *knowledge transfer* means improved access to task-relevant information through a defined message—never permanent learning.

3.2. Accounting

For a dense cache with uniform layers the stored tensor size is

$$M_{KV} = 2BLTH_{KV}d_h s, \tag{2}$$

with batch B, layers L, stored positions T, KV heads H_{KV}, head dimension d_h and s bytes per scalar. We use (2) only as a consistency check: the harness measures stored tensor bytes from the arrays themselves (which, for the packed quantized caches, include the uint32 words, the per-group scales and biases) and measures the serialized payload from the file written to disk, including the compatibility manifest. Two ratios are reported with explicit denominators,

$$CR_{\text{payload}} = \frac{\text{reference serialized payload bytes}}{\text{method serialized payload bytes}}, \qquad CR_{KV} = \frac{\text{native cache tensor bytes}}{\text{stored cache tensor and metadata bytes}}, \tag{3}$$

and a text-token ratio is never treated as a cache-position ratio.

Timing boundaries. Each trial records separate wall-clock spans with device synchronization before and after (`mx.synchronize`): sender prefill and message generation (*encode*); document prefill into the receiver cache (*doc prefill*); compression or selection; serialization and transport codec; deserialization and cache installation (*load*); fresh query prefill up to the first content token (TTFT); decoding; and, for metered methods, retrieval. For a serial workflow,

$$T_{\text{total}} = T_{\text{queue}} + T_{\text{read}} + T_{\text{sender}} + T_{\text{encode}} + T_{\text{serialize}} + T_{\text{transport}} + T_{\text{load}} + T_{\text{receiver}} + T_{\text{tools}}. \tag{4}$$

Three scenarios are assembled from the spans: *cold* (no reusable state exists: build + compress + load + receiver), *warm per-query* (the state exists: load + receiver), and *amortized over N queries*,

$$\bar{C}(N) = \frac{C_{\text{training}} + C_{\text{setup}} + C_{\text{document}}}{N} + C_{\text{query}}, \tag{5}$$

where C_{training} is zero for all training-free methods and is reported separately for the learned mapper. Components that a hosted provider does not expose are recorded as missing, not as zero.

3.3. Taxonomy

Table 2 lists every method with its representation, question-access regime, training requirement, compatibility requirement, composability across sources, and its measured status in this study. Methods that were audited but not executed appear with the reason.

3.4. Research questions and registered hypotheses

Seven research questions (RQ1–RQ7) and ten falsifiable hypotheses were registered before any held-out outcome was examined (`protocol/hypotheses.md`; reproduced in Appendix A). In brief: RQ1 compression frontier; RQ2 transfer compatibility; RQ3 whether rankings change across resource denominators; RQ4 repeated handoffs; RQ5 structured evidence versus prose; RQ6 locality

Table 2. Method taxonomy and coverage. "Measured" = executed in WP30 with real model inference; the last rows are audited methods that were not executed, with the reason.

id	representation	question_access	training	compatibility	composable	status_in_wp30
A0	none (task only)	–	none	any	–	measured
A1	raw text	blind	none	any receiver	concatenation	measured
A2	text head/tail	blind	none	any	concatenation	measured
A3	extractive sentences (BM25/centrality)	blind or aware	none	any	concatenation	measured
A4	abstractive summary	blind or aware	none (prompted sender)	any	concatenation	measured
A5	typed facts + source spans + unknowns (JSON)	blind or aware	none (prompted sender)	any	record merge	measured
A6	reference index + metered retrieval	blind index, aware retrieval	none	any + source access	shared memory	measured
A7	raw text, zstd transport codec	blind	none	any	concatenation	measured
A8	LLMLingua-2 token dropping	blind	pretrained compressor (official)	any	concatenation	measured
B0	native bf16 KV cache	blind	none	same checkpoint	no	measured
B1	serialized KV cache (manifest-checked)	blind	none	same checkpoint + revision	no	measured (process + 2-node)
B2	packed quantized KV (8/4/2 bit)	blind	none	same checkpoint	no	measured
B3	sink+recent retention (StreamingLLM-inspired)	blind	none	same checkpoint	no	measured
B4	random retention (control)	blind	none	same checkpoint	no	measured
B5	attention-informed retention (SnapKV rule)	aware (blind variant tested)	none	same checkpoint	no	measured
B6	pyramid layer budgets (PyramidKV rule)	aware	none	same checkpoint	no	measured
B9	selection then packed quantization	aware	none	same checkpoint	no	measured
C0	sender cache fed as receiver cache	blind	none	incompatible (negative control)	no	measured (control)
C2	linear KV mapper (ridge, un-rotated K/V)	blind	ridge on train caches	trained pair (Qwen 1.5B→3B)	no	measured
C3	mapper + attention selection	aware	ridge on train caches	trained pair	no	measured
–	SwaT cross-family mapper	blind	300M mapper (not public)	Granite→SmolLM	no	not run: no checkpoint/licence
–	Attention-Matching compaction	blind	none (optimization)	same model	no	not run: CUDA/vLLM code; MLX port out of scope
–	C2C fuser	blind	released fusers (Qwen3)	Qwen3 receivers	no	not run: CUDA; receiver still prefills source
–	H2O eviction	cumulative	none	same checkpoint	no	not run: requires attention scores during whole prefill
–	KIVI per-channel key quantization	blind	none	same checkpoint	no	not run: CUDA kernels; motivation confirmed by B2 Qwen result

versus moving the cache; RQ7 adaptive routing. A primary confirmatory family of six paired comparisons is Holm-corrected; all other comparisons are exploratory.

4. Methods

All methods are implemented in one Python package (`src/wp30`) on top of `mlx-lm` 0.32.2 (Apple ML Explore contributors, 2026), whose KV cache for a layer is a pair of tensors $K, V \in \mathbb{R}^{B \times H_{KV} \times T \times d_h}$ with keys stored *after* rotary position embedding (RoPE) and a logical offset that drives the positions of new tokens. We distinguish *official* code (run as released), *faithful re-implementations* (same rule and defaults, re-coded for MLX), *adapted*, *inspired* and *custom* baselines; the label of every method is in Table 2 and in `literature/method_notes.md`.

4.1. Track A: text, structured messages and evidence

A0 no message.
The receiver gets the question only (prior/guessing floor).

A1 full raw context.
The document is placed in the receiver prompt; receiver prefill is counted.

A2 head/tail truncation.
Fixed rule (half the token budget from the head, half from the tail, under the receiver tokenizer).

A3 extractive selection.
Deterministic sentence selection under a token budget: BM25 of the question over sentences (question-aware) or a centrality score weighted by the density of digits and identifiers (question-blind).

A4 abstractive summary.
Generated by the sender with a fixed prompt and a hard output cap equal to the budget; the prompt asks for verbatim identifiers, numbers, dates and negations and, in the blind regime, for everything concrete because the questions are unknown.

A5 structured evidence.
The sender emits JSON facts {entity, attribute, value, quote} plus optional `notes` and `unknowns`; each quote is resolved to a character span of the source, and unresolved quotes are counted. The receiver sees values and spans. Truncated JSON is repaired at the last complete object.

A6 references plus retrieval.
The message is A5 *without values*: a reference index {entity, attribute, span}. At answer time the receiver ranks references by BM25 of the question and fetches at most six spans (±40/160 characters) under a 2 000-byte budget; every lookup and fetched byte is recorded and added to the message bytes.

A7 lossless transport codec.
The raw text is compressed with Zstandard (level 3) for transport and decompressed at the receiver; the receiver prefill is unchanged. Codec times are measured.

A8 learned prompt compression.
LLMLingua-2 (Pan et al., 2024), official package (`llmlingua` 0.2.2) with the released `xlm-roberta-large-me` compressor, rate = budget/source tokens, run on the Metal backend of PyTorch (fp32). It is question-agnostic by construction.

A4 and A5 use the same sender checkpoint, the same budget and prompts of comparable length; both prompts were tuned on development data only. Receiver prompts are identical across methods except for the label of the material slot (Appendix D).

4.2. Track B: within-model KV compression and reuse

Every Track-B condition manipulates the receiver's real cache tensors and runs receiver inference from the resulting state. The receiver prefix is the chat-templated system prompt plus the document; the fresh input is the question and the assistant header, so that the cache boundary is identical for all conditions (*no duplicated boundary token* is a tested property).

B0 native cache.
The dense bf16 cache, branched (deep-copied) for every question.

B1 serialized reuse.
B0 serialized to `safetensors` with a compatibility manifest (checkpoint revision, config, tokenizer and chat-template hashes, prefix-token hash and count, per-layer shapes/dtypes/quantization, positional semantics, runtime versions) and loaded back; the loader rejects a mismatching manifest before any model execution.

B2 packed quantization.
`mx.quantize` along the head dimension (groups of 64 channels; 32 for 2-bit), asymmetric affine with a scale and a bias per group, packed into `uint32` words, and consumed by the quantized attention kernel—so the compressed storage is the execution representation, not a transport format that is expanded before attention. Bits $\in \{8, 4, 2\}$.

B3 sink+recent.
Keep the first four positions and the most recent positions up to the budget. Unlike StreamingLLM, retained keys keep their original positions and the logical offset is preserved (new tokens see a positional gap); it is a retention *pattern*, labelled "inspired".

B4 random.
Uniformly random positions at the same budget (seeded control).

B5 attention-informed.
SnapKV's rule: per KV head, the attention of an observation window onto prefix positions

(softmax, averaged over the window's queries and over the query heads of the GQA group), 1-D max-pooled with kernel 7, top-k per head, window always kept. The window is the fresh question (aware) or a generic 32-token instruction (blind). Window queries are computed in a separate pass over a copy of the cache, whose cost is recorded as compression time, so the main attention kernel stays fused.

B6 pyramid budgets.
PyramidKV's per-layer budget rule (verified against the KVCache-Factory code) with B5 scoring inside each layer; because layers retain different lengths, the fresh query is processed token by token (recorded).

B9 selection then quantization.
B5 followed by B2 (8-bit).

Selection produces a *sparse cache*: a subclass that stores only retained positions (per-head gathers are allowed), keeps the logical offset, and supplies an explicit boolean mask when several fresh tokens are appended (retained positions visible to all, causal among the new tokens). Correctness is established before any comparison (Section 5.3).

4.3. Track C: cross-checkpoint transfer

Level 1 of the compatibility ladder (same checkpoint, separate processes and separate nodes) is B1 executed through a TCP worker (Section **??**). Level 2 (same family, different size) uses a *linear KV mapper*: for each receiver layer ℓ_r of Qwen2.5-3B we align a sender layer $\ell_s = \mathrm{round}(\ell_r\,(L_s-1)/(L_r-1))$ of Qwen2.5-1.5B (both have two KV heads of dimension 128 and identical RoPE parameters, and they share the tokenizer, so sender and receiver caches are token-aligned), un-rotate the sender keys with their known positions, and fit by ridge regression a map from $[K_s, V_s] \in \mathbb{R}^{512}$ (plus bias) to $[K_r, V_r] \in \mathbb{R}^{512}$ on train-split caches only; mapped keys are re-rotated. The loss is reconstruction only ($\lambda_{\text{task}} = \lambda_{\text{distill}} = 0$); the mapper has $36 \times 513 \times 512 \approx 9.5$M parameters and is saved with a manifest (training documents, tokens, seconds, held-out reconstruction error). C0 feeds the sender cache into the receiver *without* mapping (shapes coincide): an incompatible-state negative control. C3 applies B5 selection to the mapped cache (mapping then compaction). This is not SwaT (Alabdulkareem et al., 2026), which trains a ∼300M mapper with generation and distillation terms across families; no SwaT checkpoint or licence was available (Appendix B). Level 3 (cross-family) was therefore not executed.

4.4. Track D: hosted receivers

A0–A6 messages built by the local sender are answered by two hosted models (Anthropic `claude-haiku-4-5`; OpenAI `gpt-5-nano`) through their documented APIs, with streaming to measure the time to the

Table 3. Checkpoints, geometry and measured prefill (ms, laptop). Cache bytes per token are the dense bf16 formula (2) with $B = 1$.

alias	checkpoint	rev.	L	H_kv	H_q	d_h	max pos	cache B/token	prefill 8k	prefill 16k
qwen3b	mlx-community/Qwen2.5-3B-Instruct-bf16	bec1ad658e38	36	2	16	128	32768	36864	968	2286
llama3b	mlx-community/Llama-3.2-3B-Instruct-bf16	6d88ba43024f	28	8	24	128	131072	114688	1007	2449
qwen1b5	mlx-community/Qwen2.5-1.5B-Instruct-bf16	4ae77cb209f0	28	2	12	128	32768	28672	494	1201

first content token, provider usage fields, documented prompt caching (Anthropic `cache_control`; OpenAI automatic prefix caching) and an estimated cost from the published price tables. The provider's cached-token field is reported as billing information, never as "KV bytes saved".

5. Experimental protocol

5.1. Models, hardware and runtimes

Table 3 lists the checkpoints (all ungated `mlx-community` conversions with bf16 weights, so weight quantization is not a confound), their attention geometry and measured prefill times. The local node is a MacBook Pro with an Apple M5 Max (18 cores, 48 GiB unified memory, macOS 27.0), Python 3.12.13, `mlx` 0.32.4, `mlx-lm` 0.32.2. Two-node experiments use two Mac Studio M3 Ultra nodes of the author's cluster (M3U96a as sender, M3U96b as receiver worker; 96 GB each; identical `mlx`/`mlx-lm` pins and checkpoint revisions) connected by a wired LAN measured at 940 Mbit/s (iperf3, four streams) with 1.2 ms RTT. Both cluster nodes concurrently hosted a CPU-light workload from another study (load average ≈ 1.8 on 32 cores); timing on the laptop was taken with no other GPU process.

5.2. Task families and splits

Synthetic private-fact ledgers.

A deterministic generator (seeded) writes a fictional capital-projects register: six projects per document with invented names, random identifiers (`XX-00X-0000`), amounts and currencies, dates, sites, suppliers, approvals or rejections, cancelled-versus-postponed statuses, timestamped revisions with an explicit precedence rule, a list of "proposals under review" with no attributes (distractors with similar names), and boilerplate filler to reach 2 000, 8 000 or 16 000 target tokens. Fifteen questions per document cover single-fact retrieval at varying positions, exact copying, multi-hop joins, negation, revision/precedence, temporal order, aggregation, unit conversion and two unanswerable questions (correct action: `unknown`). An evaluator-only fact ledger (62 atomic facts per document) and the answer key are stored apart from the agent-visible text. The facts are not inferable from pre-training. A procedurally generated ledger is not a real

Table 4. Datasets, independent units (documents), questions and median length by split.

family	split	documents	questions	median_chars
exec	dev	30	30	7847
exec	test	40	40	7817
exec	train	20	20	7866
public_2wikimqa	dev	56	56	25,419
public_2wikimqa	test	144	144	25,488
public_hotpotqa	dev	61	61	65,814
public_hotpotqa	test	139	139	60,944
public_qasper	dev	50	50	19,844
public_qasper	test	150	150	23,154
synthetic	dev	30	450	29,225
synthetic	test	40	600	29,203
synthetic	train	40	600	29,196

filing.

Public long-document QA.

LongBench v1 (Bai et al., 2024) subsets QASPER (Dasigi et al., 2021) and 2WikiMultihopQA (Ho et al., 2020) (raw `data.zip`; the dataset script is no longer supported by the `datasets` library), official F1 with the LongBench normalization. Documents are used in full. We run *subsets* (the first 40 test documents of each subset in manifest order) and publish the identifiers. These texts may appear in pre-training corpora; they serve external validity, not the contamination-free test of communication dependence.

Executable tasks.

A generated engineering ticket specifies six business rules for a function `net_revenue(rows)` together with a buggy implementation missing two rules; the agent must output the corrected function, which is executed in an isolated `python -I` subprocess against five hidden test cases (exact float comparison). Agents never see the tests.

Splits are by document (train/dev/test seeds for the generated families; a hash rule for the public subsets), with exact-duplicate checks across splits (none found). Train is used for mapper fitting and prompt tuning, dev for method selection and router fitting, test for the locked campaign (Table 4). Answer keys live in `data/generated/keys` and are read only by the scorer; a leakage test asserts that agent-visible files contain no answers, facts or tests.

5.3. Cache correctness before performance

Twelve tests run a real 1.5B checkpoint (Appendix C). Identity round trip (serialize, load) and 100 % retention through the sparse adapter reproduce the native next-token logits *exactly*. Cached-prefix-plus-query and full prefill agree within a registered distributional tolerance: MLX kernels are not bitwise deterministic across prefill chunkings even in float32 (we measured a maximum logit

difference of 0.10 between a one-chunk and a token-by-token prefill, with $\max|\Delta p| \approx 7 \times 10^{-7}$ and $\mathrm{KL} \approx 3 \times 10^{-8}$), so equivalence is defined as $\mathrm{KL} < 10^{-5}$, $\max|\Delta p| < 10^{-4}$ and identical greedy top-1 in float32 ($\mathrm{KL} < 10^{-3}$ in bf16). The sparse-cache mask path agrees with token-by-token appends; preserving original positions differs measurably from relabelling them; packed storage has the intended bit width (uint32 words, ratios 1.9/3.6/6.4 for 8/4/2 bits on a 412-token prefix including scales and biases); the loader rejects a wrong prefix hash or a different checkpoint; branches do not share mutable state; and the no-message control does not know the private answer.

5.4. Pilot, freeze and campaign

A development pilot (`E1_pilot`: 16 documents, 31 conditions, two model pairs) calibrated budgets, revealed implementation failures and estimated variance and run time. The confirmatory protocol was then frozen (`protocol/frozen.json`, hash `unfrozen`) with the final hypotheses, prompts, budgets, scorers and analysis plan, before any test-split item was evaluated. The core single-handoff campaign (`E1_core`) covers 160 test documents (40 per family), 22 conditions and two model pairs (13 600 scheduled trials); the handoff, fan-in, transfer, network and hosted campaigns are described with their results.

5.5. Statistics

The unit is a (document, question, condition, pair) trial; documents are the independent clusters. Means and paired differences on identical instances are reported with document-level cluster-bootstrap 95 % intervals (2 000 replicates). Six registered primary comparisons are Holm-corrected; everything else is exploratory. Quality is accuracy (synthetic, executable) or F1 (public); failed or invalid attempts count as 0 in the operational quality and are also tabulated by category. Latency is reported as p50/p95 with sample counts, from the spans defined in Section 3.

6. Results

To be written from results.

7. Adaptive

To be written from results.

8. Limitations

To be written from results.

9. Reproducibility

To be written from results.

10. Conclusion

To be written from results.

References

Alabdulkareem, Abdulrahman, Isabel Tenison, Jun Zhang, Yiyang Zhu, Boris Katz, and Dan Gutfreund, "SwaT: Cross-Family KV Cache Transfer between Language Models," 2026. arXiv preprint, v1 (October 2026); repository Ar-Kareem/SwaT, no released checkpoint or licence (verified 2026-10-29).

Anthropic, "Prompt caching," https://platform.claude.com/docs/en/build-with-claude/prompt-caching 2026. accessed 2026-10-29.

Apple ML Explore contributors, "mlx-lm: Run LLMs with MLX," https://github.com/ml-explore/mlx-lm 2026. v0.32.2 (16 October 2026), MIT.

Bai, Yushi, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li, "LongBench: A Bilingual, Multitask Benchmark for Long Context Understanding," in "Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)" 2024, pp. 3119–3137.

Cai, Zefan, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao, "PyramidKV: Dynamic KV Cache Compression based on Pyramidal Information Funneling," 2024. v4 (May 2025); code Zefan-Cai/KVCache-Factory, MIT.

Dasigi, Pradeep, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A. Smith, and Matt Gardner, "A Dataset of Information-Seeking Questions and Answers Anchored in Research Papers," 2021. NAACL 2021; QASPER, CC BY 4.0.

Fu, Tianyu et al., "Cache-to-Cache: Direct Semantic Communication Between Large Language Models," in "International Conference on Learning Representations (ICLR)" 2026. arXiv:2510.03215; code thu-nics/C2C, Apache-2.0; fuser checkpoints nics-efc/C2C_Fuser.

Ho, Xanh, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa, "Constructing A Multi-hop QA Dataset for Comprehensive Evaluation of Reasoning Steps," in "Proceedings of the 28th International Conference on Computational Linguistics (COLING)" 2020.

Hsieh, Cheng-Ping, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg, "RULER: What's the Real Context Size of Your Long-Context Language Models?," in "First Conference on Language Modeling (COLM)" 2024. arXiv:2404.06654.

Hugging Face, "KV cache strategies," https://huggingface.co/docs/transformers/en/kv_cache 2026. Transformers v5 documentation, accessed 2026-10-29.

Li, Yuhong, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen, "SnapKV: LLM Knows What You are Looking for Before Generation," in "Advances in Neural Information Processing Systems (NeurIPS)" 2024. arXiv:2404.14469; code FasterDecoding/SnapKV.

Liu, Zirui, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu, "KIVI: A Tuning-Free Asymmetric 2bit Quantization for KV Cache," in "International Conference on Machine Learning (ICML)" 2024. arXiv:2402.02750.

Pan, Zhuoshi, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, H. Vicky Zhao, Lili Qiu, and Dongmei Zhang, "LLMLingua-2: Data Distillation for Efficient and Faithful Task-Agnostic Prompt Compression," in "Findings of the Association for Computational Linguistics: ACL 2024" 2024. arXiv:2403.12968.

Xiao, Guangxuan, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis, "Efficient Streaming Language Models with Attention Sinks," in "International Conference on Learning Representations (ICLR)" 2024. arXiv:2309.17453.

Zhang, Zhenyu, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen, "H_2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models," in "Advances in Neural Information Processing Systems (NeurIPS)" 2023. arXiv:2306.14048.

Zou, Jiaru et al., "LatentMAS: Latent Collaboration in Multi-Agent Systems," in "International Conference on Machine Learning (ICML)" 2026. arXiv:2511.20639; code Gen-Verse/LatentMAS, Apache-2.0.

Zweiger, Adam, Yilun Fu, Han Guo, and Yoon Kim, "Fast KV Compaction via Attention Matching," 2026. arXiv preprint, v2 (May 2026); code adamzweiger/compaction, MIT.

Appendices

A. Registered hypotheses

```
# WP30 -- Research questions and falsifiable hypotheses (registered before any held-out outcome was exam

Registered: 2026-10-29 (Phase 1). Amendments only as dated entries at the bottom.
Test-split outcomes had **not** been examined at registration (no test split existed yet; only the E0 sm
hand-written 3-sentence memo had been executed, see `runs/logs/`).

## Research questions

- **RQ1 (compression frontier)** How do text shortening (A2-A4, A8), packed KV quantization (B2), token
  learned compaction (B7, if run) trade task quality against *measured* serialized bytes and resident ca
- **RQ2 (transfer compatibility)** How much performance is preserved by same-checkpoint reuse (B1, cross
  versus learned mappings between distinct checkpoints (Track C), relative to the receiver's own full-co
- **RQ3 (resource accounting)** Do rankings of methods under a token budget change when the budget is by
  peak memory or amortized cost over N reuses?
- **RQ4 (repeated handoffs)** How do correct-fact retention, unsupported additions and task accuracy evo
  for prose summaries vs structured evidence, in closed vs repairable chains?
- **RQ5 (evidence)** Under an equal byte budget, does structured evidence with source spans (A5) retain
  numbers, dates, negations) better than an abstractive summary (A4)? Does metered retrieval (A6) repair
- **RQ6 (locality)** When is running the next query where the cache resides cheaper than moving the cach
  as a function of context length, payload bytes, measured bandwidth and queue delay?
- **RQ7 (adaptive policy)** Can a router fitted on development data improve the held-out quality/cost fr
  development-selected static method, once routing overhead and fallbacks are counted?

## Hypotheses (each may be rejected)

- **H1** Lossy compression degrades exact identifiers, decimal numbers, negation and temporal order more
  on the synthetic private-fact family, the accuracy drop from A1 to A4 at a 512-token budget is larger
  for `exact_copy`, `negation`, and `revision` question types than for `single_fact` questions.
- **H2** Packed KV quantization is model-dependent: at 4 bits/group 64, Llama-3.2-3B retains >= 95 % of
  Qwen2.5-3B loses >= 20 percentage points (motivated by E0 smoke: Qwen2.5-1.5B produced garbage at 4-bi
  mlx_lm path and ours; Llama-3.2-3B did not). 8-bit retains >= 97 % for both.
- **H3** Sink+recent eviction (B3) at <= 1/4 retention is worse than a *question-aware* attention-inform
  at equal retained positions, but **not** reliably better than random selection (B4) on questions whose
  middle of the document.
- **H4** Serialized cache bytes exceed raw UTF-8 document bytes by >= 2 orders of magnitude for 3B model
  wins on total latency when (a) the receiver's prefill cost exceeds transport time, which on the measur
  above a break-even length we will estimate (prediction: several thousand tokens), or (b) the cache is
- **H5** Repeated prose handoffs lose facts monotonically; after 4 handoffs the retained-correct-fact ra
  70 % while A5 (structured, explicit unknowns) stays above 85 %; unsupported additions appear mainly in
```

```
- **H6** Metered retrieval (A6, repairable chains) restores most of the lost exact facts but adds fetche
  the message itself.
- **H7** Question-aware compression dominates question-blind compression at equal budget for *all* lossy
  largest for attention-based selection (which is query-dependent by construction).
- **H8** A lossless transport codec (A7, zstd) on text yields >= 2x byte savings with < 5 ms overhead bu
  latency or quality; on bf16 KV tensors the codec yields < 1.3x (high-entropy payload).
- **H9** The adaptive router (E8) will not reliably beat the best development-selected static method on
  overhead is included (we expect a null or small, uncertain gain).
- **H10** Cross-family cache transfer without a trained mapper is not possible; with a mapper trained un
  quality is below the receiver's own A4 summary baseline (we expect learned transfer to lose to text at

## Primary endpoints (frozen at Phase 7 together with the matrix)

Per task family: exact-match / F1 (LongBench normalization) or numeric correctness with tolerance; opera
serialized payload bytes; end-to-end wall time (defined boundaries, S6.1 of the brief); peak MLX memory.

## Primary confirmatory comparison family (Holm-corrected; everything else exploratory)

1. A5 vs A4 at matched byte budget (synthetic, question-blind) -- accuracy.
2. B2-8bit vs B0 -- accuracy (both models).
3. B2-4bit vs B0 -- accuracy (Qwen vs Llama interaction stated as H2).
4. B5-aware vs B3 at 1/4 retention -- accuracy.
5. A4 chain vs A5 chain at 4 handoffs -- fact retention.
6. Router vs best static -- held-out accuracy at equal or lower cost.

## Amendments

(none yet)
```

B. Source audit and unexecuted methods

Thirty-four sources were verified on 2026-10-29 from their arXiv pages, repositories and documentation (`literature/sources.csv`; notes in `literature/audit_group{1,2,3}.md`). Table 5 summarizes the core methods.

C. Correctness tests

`tests/test_cache_correctness.py` (12 tests, all passing on Qwen2.5-1.5B-Instruct-bf16): prefix equivalence in float32 and bf16 (distributional tolerance of Section 5.3, identical greedy continuation); no duplicated boundary token; identity round trip with exact logits and measured bytes

Table 5. Audited core methods: availability of code and checkpoints, backend, question access, and WP30 status.

Method	Code	Checkpoint	Backend	Question access	WP30
SwaT (2610.15095)	yes (no licence)	none	torch/CUDA	blind	not run
Attention Matching (2602.16284)	MIT	n/a (optimization)	vLLM 0.11 / torch	blind	not run (MLX port out of scope)
C2C (2510.03215)	Apache-2.0	Qwen3 fusers	torch/CUDA	blind	not run
LatentMAS (2511.20639)	Apache-2.0	n/a	HF/vLLM, CUDA	blind	not run
KIVI (2402.02750)	MIT	n/a	CUDA kernels	n/a	motivation tested via B2
SnapKV (2404.14469)	Apache-2.0	n/a	torch	aware	faithful re-implementation (B5)
PyramidKV (2406.02069)	MIT	n/a	torch	aware	budget rule re-implemented (B6)
H_2O (2306.14048)	no LICENSE file	n/a	torch	cumulative	not run
StreamingLLM (2309.17453)	MIT	n/a	torch	blind	retention pattern only (B3)
LLMLingua-2 (2403.12968)	MIT	yes (HF)	torch (MPS ok)	blind	official (A8)

equal to (2); loader rejection of a wrong prefix hash and of a different checkpoint; 100 % retention exact (shared and per-head indices); sparse mask path versus token-by-token append; preserved versus relabelled positions differ; packed quantization bit widths and execution for 8/4/2 bits; no shared mutable state across branches; attention selection keeps the window; no-message control. `tests/test_leakage.py` (3) and `tests/test_scoring_and_data.py` (5) cover evaluator isolation, scorer behaviour and generator determinism.

D. Prompts

```
"""Prompt templates (frozen at Phase 7). Receiver prompts are identical across methods except for the me
SYSTEM_RECEIVER = ("You are a careful assistant. Answer the question using ONLY the provided material. "
                   "If the material does not contain the answer, reply exactly: unknown.")
SYSTEM_RECEIVER_NOMSG = ("You are a careful assistant. Answer the question. If you do not know, reply ex

def receiver_user_text(material_label: str, material: str, question: str, family: str) -> str:
    if family == "exec":
        return f"{material_label}:\\n{material}\\n\\nTask: {question}"
    fmt = "Give a short answer (a few words or a number), with no explanation."
    return f"{material_label}:\\n{material}\\n\\nQuestion: {question}\\n{fmt}"

def receiver_question_only(question: str, family: str) -> str:
    if family == "exec":
        return f"Task: {question}"
    return f"Question: {question}\\nGive a short answer (a few words or a number), with no explanation."

# Sender prompts (question-blind: no question shown; question-aware: question appended)
SUMMARY_SYSTEM = "You are a precise note-taker. You write faithful summaries that preserve exact names,
def summary_user(doc: str, budget_tokens: int, question: str | None) -> str:
    q = f"\\nThe reader will need to answer this question: {question}\\n" if question else "\\nThe reade
    return (f"Summarize the following document in at most {budget_tokens} tokens (about {int(budget_toke
```

```
            f"numbers, dates and names exactly as written. Mark explicitly what is NOT stated if relevan

EVIDENCE_SYSTEM = ("You extract structured evidence. Output ONLY a JSON object with a key \\"facts\\": a
                   "{\\"entity\\": str, \\"attribute\\": str, \\"value\\": str, \\"quote\\": str} where
                   "supporting the fact, and optionally a key \\"notes\\": a short string for anything i
                   "and a key \\"unknowns\\": list of things the document explicitly leaves unstated. Co
def evidence_user(doc: str, budget_tokens: int, question: str | None) -> str:
    q = f"\\nThe reader will need to answer this question: {question}\\n" if question else "\\nThe reade
    return f"Extract structured evidence from the document. Keep the JSON under {budget_tokens} tokens.
```

E. Pseudocode

Algorithm 1 Trial scheduler (resumable, append-only)

load config, manifest, resources; $done \leftarrow$ trial keys with status success in `runs/raw/<exp>.jsonl`
for document d in selected items **do**
  for model pair (S,R) **do**
    for condition c **do**
      for seed **do**
        $P \leftarrow$ questions of d whose trial key $\notin done$; **continue** if $P = \emptyset$
        if c is question-blind **then** build message/cache m once for d (reuse count $= |P|$)
        end if
        for question $q \in P$ **do** build m if question-aware; run receiver; score with evaluator-only key; append validated record
        end for
      end for
    end for
  end for
end for

Algorithm 2 Cache handoff pipeline (Track B/C)

pre $\leftarrow$ chat tokens of (system, document) that are a true prefix of the full (system, document, question) prompt
$C \leftarrow$ receiver.prefill(pre) ▷ dense bf16; measured doc prefill
$C' \leftarrow$ compress(C): quantize / select positions (sparse cache with logical offset $|\text{pre}|$) / map from sender
payload $\leftarrow$ serialize(C', manifest); optionally codec
at receiver: check manifest; deserialize; install; prefill fresh query tokens with retained-positions mask; decode greedily

F. Per-attempt record schema

Each attempt is one JSON line validated against `wp30-attempt-1`: identifiers (study, experiment, trial, attempt, run group, protocol and config hashes, git commit, timestamp), task (family, document id and hash, question id, split, cluster), models (sender and receiver revisions, tokenizer hashes,

Algorithm 3 Evidence retrieval (A6) and repairable chains

refs ← A5 facts without values; rank refs by BM25(question, entity+attribute)
for ref in ranked order while lookups $<$ max and bytes $<$ budget **do** fetch source span $\pm$ context; count bytes and lookups
end for
chains: for codes mentioned in the previous message, fetch source sentences containing the code under the same meters

Algorithm 4 Adaptive policy (E8)

on dev: for each (family, context bin, pair, access) and action a: $\hat{q}_a$, lower bootstrap bound $\underline{q}_a$, mean cost $\hat{c}_a$
feasible $\leftarrow \{a : \underline{q}_a \geq \tau \max_{a'} \underline{q}_{a'}\}$; choose $\arg\min_{a \in \text{feasible}} \hat{c}_a$; fallback to the dev-best static action
freeze; on test: look up the cell (nearest bin fallback counted), apply the chosen action, add router time

backend versions, hardware id, device state, weight precision), method (id, fidelity, compression config with realized values, question access, topology, handoff index, reuse count, cache hit, compatibility hash, seed), sizes (source tokens under both tokenizers, query and message tokens, UTF-8 bytes, cache tensor bytes, serialized payload bytes, side information, fetched bytes, transport bytes), timing spans and TTFT, usage and provider cached usage, memory (process RSS, MLX active/peak, prefill and decode peaks, mapper memory), cost with source, prediction and evidence artifacts, key reference, scores, status, error code, retry link, log path, payload and output hashes.

G. Additional tables